Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Системы искусственного интеллекта
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Г.М. Цибульский
подпись инициалы, фамилия
«___» _____ 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

«Применение искусственных нейронных сетей к задаче распознавания текста на изображениях сложных графических сцен»
тема

09.04.01 «Информатика и вычислительная техника»
код и наименование направления

09.04.01.10 «Интеллектуальные информационные системы»
код и наименование магистерской программы

Научный руководитель _____ _____ А.В. Пятаева
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник _____ С.А. Генза
подпись, дата инициалы, фамилия

Рецензент _____ _____ В.В. Вдовенко
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2019

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Системы искусственного интеллекта
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Г.М. Цибульский
подпись инициалы, фамилия
«___» _____ 2019 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме магистерской диссертации

Студенту Генза Сергею Андреевичу

Группа: КИ17-01-10М. Направление: 09.04.01 «Информатика и вычислительная техника».

Тема выпускной квалификационной работы:

«Применение искусственных нейронных сетей к задаче распознавания текста на изображениях сложных графических сцен»

Утверждена приказом по университету № _____ от _____

Руководитель ВКР: А.В. Пятаева, канд.техн. наук, доц., доцент кафедры «Системы искусственного интеллекта» ИКИТ СФУ.

Исходные данные для ВКР:

–научная, справочная и периодическая литература;

–высокоуровневый язык программирования Python, библиотека глубокого обучения нейронных сетей Tensorflow;

Перечень разделов ВКР:

– обзор методов распознавания текста на сложных графических сценах;

– подход с применением CRNN-архитектуры нейронных сетей;

– эксперимент.

Перечень графического материала:

– иллюстрация «Графики изменения функции потерь, расстояния Левенштейна и точности распознавания»;

– иллюстрация «Результаты эксперимента и сравнение с другими подходами».

Руководитель ВКР _____ А.В. Пятаева
подпись

Задание принял к исполнению _____ С.А. Генза
подпись

« ___ » _____ 20__ г.

График

выполнения выпускной квалификационной работы студентом направления 09.04.01 «Информатика и вычислительная техника», магистерской программы 09.04.01.10 «Интеллектуальные информационные системы» приведен в таблице 1.

Таблица 1 – График выполнения этапов ВКР

Наименование этапа	Срок выполнения этапа	Результат выполнения этапа	Примечание руководителя
Ознакомление с целью и задачами работы	12.12.2017	Краткое эссе по теме магистерской диссертации	Выполнено
Сбор литературных источников	24.01.2018	Список литературных источников, подходящих к теме диссертации. Понимание предмета исследования	Выполнено
Анализ выбранных литературных источников	25.02.2018	Реферат по теме магистерской диссертации	Выполнено
Уточнение и обоснование актуальности цели и задач исследования	15.03.2018	Окончательная формулировка цели и задач исследования	Выполнено
Формирование обзорной части исследования	18.05.2018	Обзорная часть исследования	Выполнено
Исследование методов распознавания текста на изображениях сложных графических сцен	25.01.2019	Результаты работы исследованных методов, представленные в виде таблицы. Выводы к ним.	Выполнено
Программная реализация подхода с применением CRNN-архитектуры к решению задачи распознавания текста на изображениях сложных графических сцен	04.04.2019	Программа на ЭВМ с реализацией CRNN-архитектуры нейронных сетей. Проведение эксперимента, анализ результатов и выводы по ним.	Выполнено
Защита ВКР	10.07.2019	Доклад и презентация по результатам магистерской диссертации	Выполнено

Студент _____ С.А. Генза
подпись

Научный руководитель _____ к.т.н., доцент А.В. Пятаева
подпись

АННОТАЦИЯ

В работе проведен обзор и анализ существующих методов локализации и распознавания текста на изображениях сложных графических сцен. Предложен подход к распознаванию локализованного текста, базирующийся на сочетании рекуррентных, сверточных нейронных сетей (CRNN) и алгоритма CTC-loss. Данная архитектура нейронных сетей реализована с помощью языка программирования Python. Проведен эксперимент на двух наборах данных, по результатам которого были построены графики изменения в течение обучения функции потерь, расстояния Левенштейна и точности распознавания на тренировочном и двух тестовых наборах данных. Распознавание проводилось двумя способами: с использованием дополнительного словаря фиксированного размера и без его использования. Составлена таблица значений точности распознавания текста на двух наборах данных для подхода, предлагаемого в данной работе и других подходов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сложные графические сцены – это изображения, характеризующиеся отсутствием четких критериев отличия фона от текста, неоднородностью фона, большой вероятностью разнообразных искажений и зашумления [1]. Такой текст может быть переменного качества, различного шрифта, наклона, формы, толщины и текстуры. Примерами таких сцен могут являться фотографии уличных вывесок, кадры из кинофильмов с субтитрами, данные в системах навигации роботов и т.п.

Системы распознавания текста (т.н. OCR-системы), существующие на данный момент, ориентированы на бумажные документы, в которых текст располагается на однородном контрастном фоне и легко подвергается сегментации и бинаризации. Распознавание текста на сложных графических сценах, в отличие от распознавания текста на бумажных документах, осложняется невозможностью явной локализации (detection) текста. Система не имеет априорной информации о том, где и как на изображении расположен текст, и локализация текста – это отдельный этап работы таких систем. Также данная задача примечательна тем, что на данных изображениях часто встречаются специфические слова (названия компаний, марки автомобилей, жаргонизмы в тексте рекламы и т.д.). По этой причине осложняется использование методов с заранее заданным словарем, содержащим все возможные вариации распознаваемых слов.

Известны уже существующие системы локализации текста, используемые в распознавании номерных знаков автомобилей и меток даты и времени на фотографиях. Решение таких задач облегчается тем, что распознаваемый текст имеет фиксированную длину и алфавит (набор возможных символов), а также находится на контрастном фоне и может быть обработан алгоритмами сегментации. В настоящей работе решается задача распознавания уже локализованного текста.

Целями и задачами магистерской диссертации являются:

- обзор и анализ существующих на данный момент методов локализации и распознавания текста на изображениях сложных графических сцен;
- описание и исследование подхода с применением CRNN архитектуры нейронных сетей к распознаванию локализованного на изображении текста;
- проведение эксперимента с реализацией CRNN архитектуры нейронных сетей и сравнение результатов на двух наборах данных с другими подходами.

1 Обзор методов распознавания текста на сложных графических сценах

Оптическое распознавание символов (англ. Optical Character Recognition, OCR) – это механический или электронный перевод изображений рукописного, машинописного и печатного в текстовые данные. Данная технология позволяет редактировать текст, осуществлять машинный перевод, семантическую обработку текста, а также значительно экономить память при хранении документов.

Задача распознавания текста не ограничивается обработкой бумажных документов, актуальна также и задача распознавания текста на сложных графических сценах, изображенных на фотографиях и кадрах видеосъемки.

Основное отличие – это расположение текста на изображении. OCR-системы имеют априорное знание о расположении текста в документе. В тексте выделяются отдельные строки, слова и символы, которые после предобработки подаются на вход алгоритму классификации.

Примерами сложных графических сцен могут являться фотографии уличных вывесок, кадры из кинофильмов с субтитрами, данные в системах навигации роботов и т.п. Решение задачи распознавания подобного текста позволит использовать такие системы на мобильных платформах, видео- и фотокамерах, в поисковых системах и многом другом.

Локализовать текст на изображении со сценой гораздо сложнее. Помимо самого текста на изображении могут присутствовать другие объекты – деревья, автомобили, люди и т.п, и задача локализации текста – отделение текста на изображении от остальных объектов.

Распознавание текста на графической сцене делится на три этапа:

- локализация текста на изображении;
- распознавание текста на локализованном изображении;
- проверка найденного слова по словарю.

Рисунок 1 – Примеры изображений сложных графических сцен

Рисунок 2 – Пример некорректной работы алгоритма бинаризации

1.1 Локализация текста

Проведем обзор методов, предлагаемых в научно-технической литературе для решения задачи локализации текста на изображениях сложных графических сцен.

Работа Y. Kunishige и соавторов [2] опирается на идею использования т.н. «контекста окружения» (environmental context). Основной принцип заключается в использовании информации о том, что окружает область-«кандидата». Предлагается анализировать тот фон, на котором находится регион изображения, возможно, являющийся текстовым. Идея базируется на эмпирическом

предположении, что вероятность наличия текста, например, на травяном покрове или на небе - низка.

Второй материал того же автора [3], а также статья авторского коллектива Lao, Du и др. [4] посвящены разработке идеи так называемых feature-точек.

В работе [5] описывается идея автоматической генерации признаков, используемых для распознавания. Основная идея заключается в том, что признаки для алгоритма классификации можно не составлять вручную, а создавать их с помощью машинного обучения. В качестве базы для обучения предлагается использовать так называемые патчи, имеющие размер 8x8 пикселов. Дальнейшая обработка изображения сводится к вычислению обученных признаков на интересующих областях изображения.

Основная идея ряда работ (например, [6]) заключается в том, что буквы и слова на изображении, как правило, имеют постоянную толщину штриха. Поэтому для выявления таких объектов, по мнению авторов, перспективно использовать алгоритм SWT (Stroke Width Transform). Ширина штриха в букве должна быть приблизительно постоянна, равно как и буквы в одном слове должны иметь похожую ширину. Признак тем более полезен, так как использовать его возможно как в качестве одного из признаков при классификации регионов, так и в качестве признака при «собирании» регионов в слова.

Границы символов в рамках описанного подхода могут вычисляться, например, с помощью Canny Egde Detector. Затем в направлении градиента находится парная граничная точка. Если градиент в этой точке примерно коллинеарен градиенту в первой точке, то все точки между ними заполняются значением найденной ширины.

Затем, при втором проходе по найденным на первом проходе отрезкам вычисленное значение ширины ограничивается медианным значением вдоль отрезка. При этом следует помнить, что предложенный метод потребует

определённых дополнительных вычислительных затрат для борьбы с ошибками на углах и некоторыми другими специфическими для него эффектами.

Рисунок 3 – Визуализация результатов метода локализации текста, предложенного в статье [5]

В работе [4] к аналогичной задаче предложен несколько иной подход. Там также решается задача поиска точечного текста, но для обнаружения точек, составляющих буквы, применяется достаточно хорошо известный алгоритм FAST. Затем производится эвристическая фильтрация ложных кандидатов, объединение точек в буквы, букв - в слова, после чего применяется классификатор SVM (Support Vector Machine) для детектирования текстовых областей.

Авторы работы [7] предлагают использовать сверточные нейронные сети для решения данной задачи. При обучении на вход такой сети подается

изображение всей графической сцены, а в качестве ожидаемого выхода используется локализованный текст этого же изображения.

1.2 Распознавание текста

Распознавание текста на сложной графической сцене представляет собой процесс декодирования полученного на этапе локализации изображения в последовательность символов.

Такие методы условно делятся на три категории [8]:

- методы на основе символов(character-based) [9, 10];
- методы на основе слов(word-based) [11];
- методы на основе последовательностей(sequence-based) [12].

Методы на основе символов(character-based) сначала, как правило, локализуют отдельные символы, после чего классифицируют их и объединяют в слова.

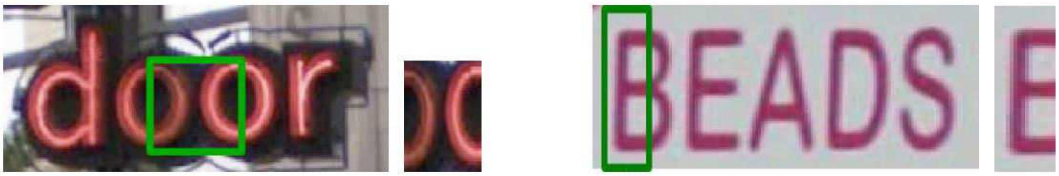

Рисунок 4 – Пример некорректной работы алгоритма локализации символов

В работе [13] предлагается следующий подход к распознаванию последовательности цифр: глубокой сверточной сетью из входного изображения X размером 128x128x3 извлекается вектор признаков $H \in R^{4096}$, которые далее передаются 6-ти независимым (без weight sharing) слоям с активационной функцией softmax для вычисления распределения вероятностей $P(S_1|H), \ldots, P(S_5|H)$ каждого символа в последовательности и вероятности $P(L|H)$ для определения длины последовательности. Примеры изображений с номерными знаками домов, распознаваемыми данной системой приведены на рисунке 5, визуализация вычислительного графа данной сети - на рисунке 7.

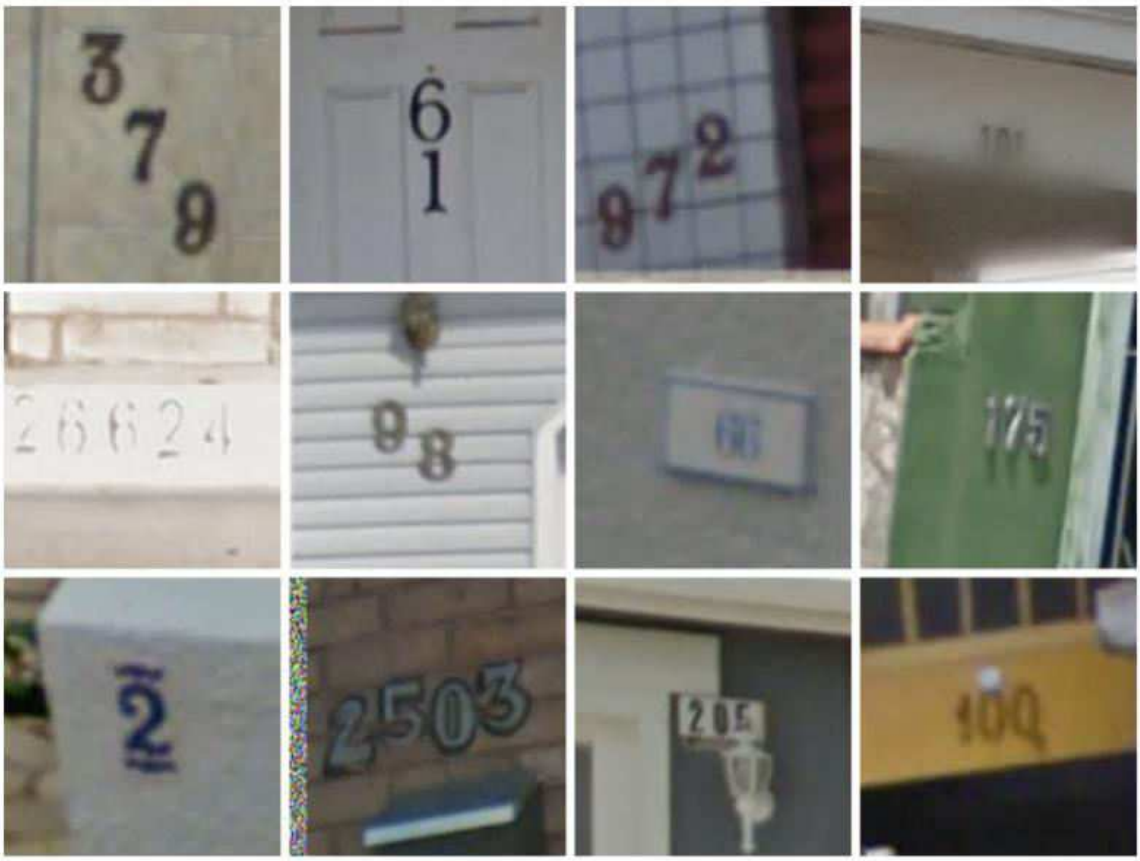

Рисунок 5 – Примеры изображений с номерными знаками домов, распознаваемыми данной системой

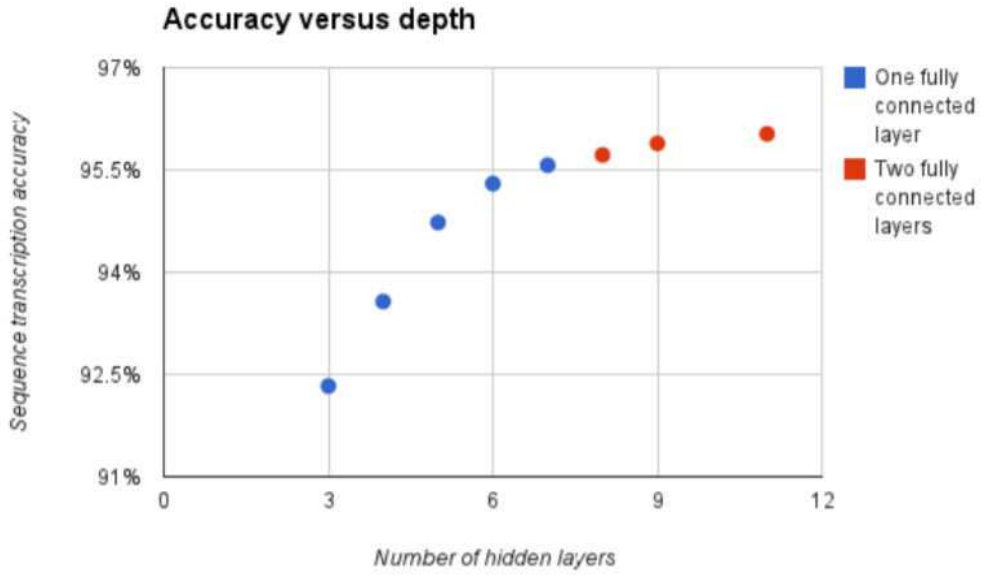

Рисунок 6 – График зависимости точности распознавания от глубины сети

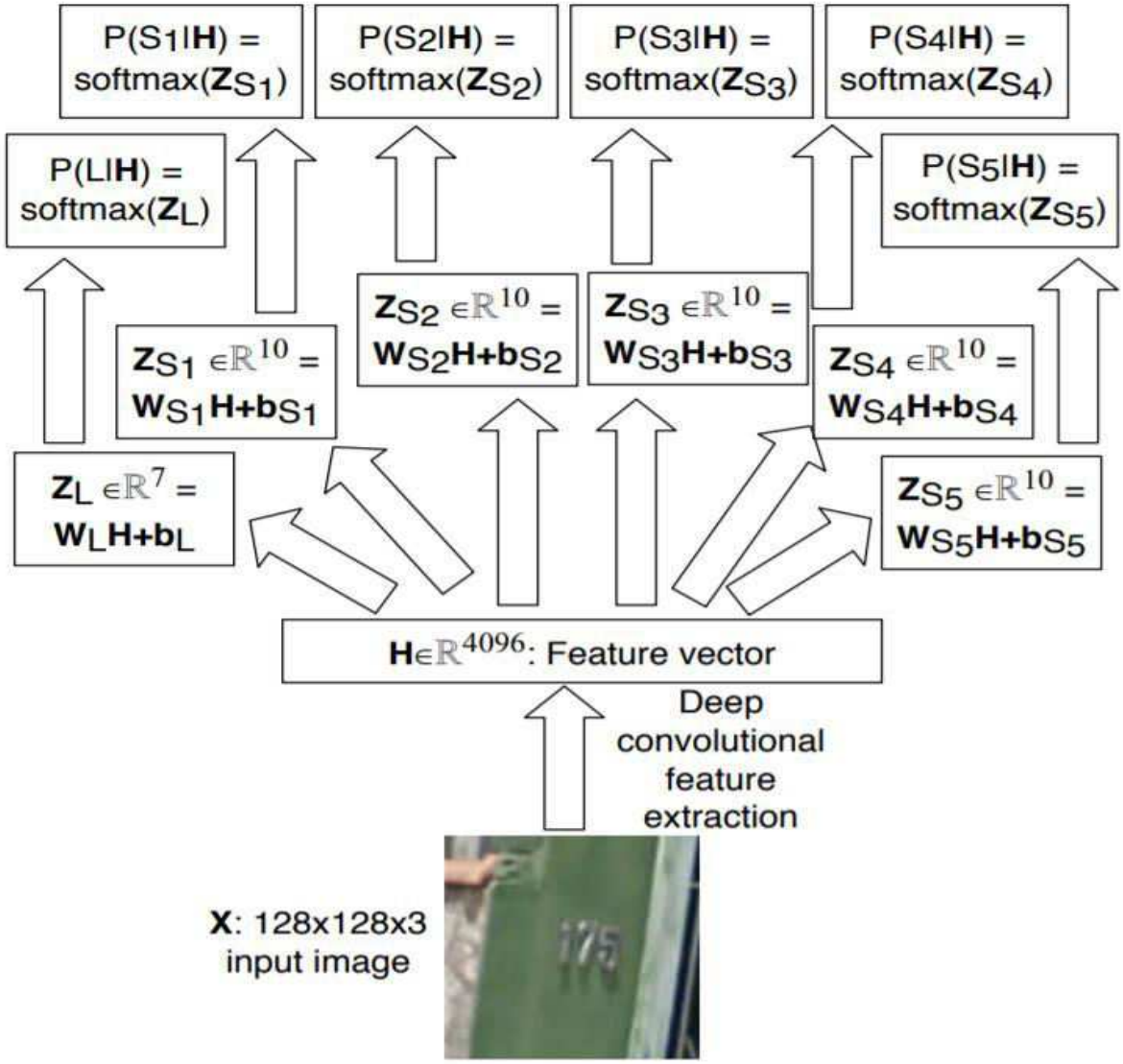

Рисунок 7 – Визуализация вычислительного графа нейронной сети из статьи [13]

Методы на основе слов(word-based) не локализуют отдельные символы, а распознают слова целиком [11]. Max Jaderberg, в совместной работе с другими авторами [12], предлагает использовать сверточные нейронные сети с заранее определенным словарем слов. Выходной слой данной сети представляет собой вектор распределения вероятностей размером 1x1xN, где N – число слов в словаре. Данный подход прост в реализации, но недостаточно гибок на практике при распознавании слов, отсутствующих в словаре.

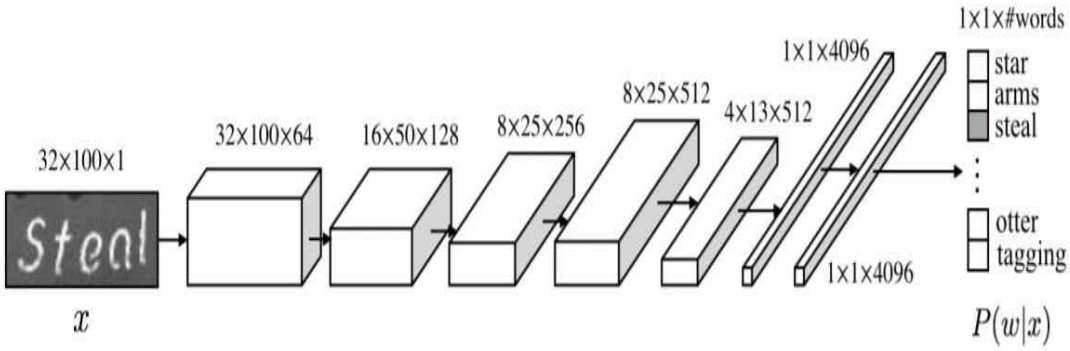

Рисунок 8 – Архитектура сверточной нейронной сети, предлагаемой в статье [12]

Методы на основе последовательностей(sequence-based) базируются на представлении входного изображения в виде заданной последовательности и её декодировании [14]. К этим методам относится и использование архитектуры нейронных сетей, описываемой в данной работе(CRNN+CTC-loss) [15].

В случаях, когда применение вышеописанных методов не обеспечивает требуемую точность распознавания, применяется проверка распознанного слова по словарю с целью повышения точности классификации. При таком подходе производится посимвольная проверка каждого распознанного слова и сопоставление его со словами в заранее составленном словаре.

В статье о CRNN[15] авторами предлагается сочетать сверточные и рекуррентные сети, а в качестве функции потерь использовать CTC loss. Данный подход на момент публикации показывал state-of-the-art результаты в решении данной задачи, так же как и его последующие модификации. На рисунке 9 приведена визуализация CRNN.

В работе [16] предлагается использовать нейронные сети с вниманием (attention) перед рекуррентными слоями. Архитектура такой сети показана на рисунке 10.

Авторами статьи [17] предлагается сперва при помощи скользящего окна выделять на исходном изображении n участков с последующим выделением

признаков сверточными слоями, классификацией и передачей алгоритму CTC-loss. Архитектура нейронной сети представлена на рисунке 11.

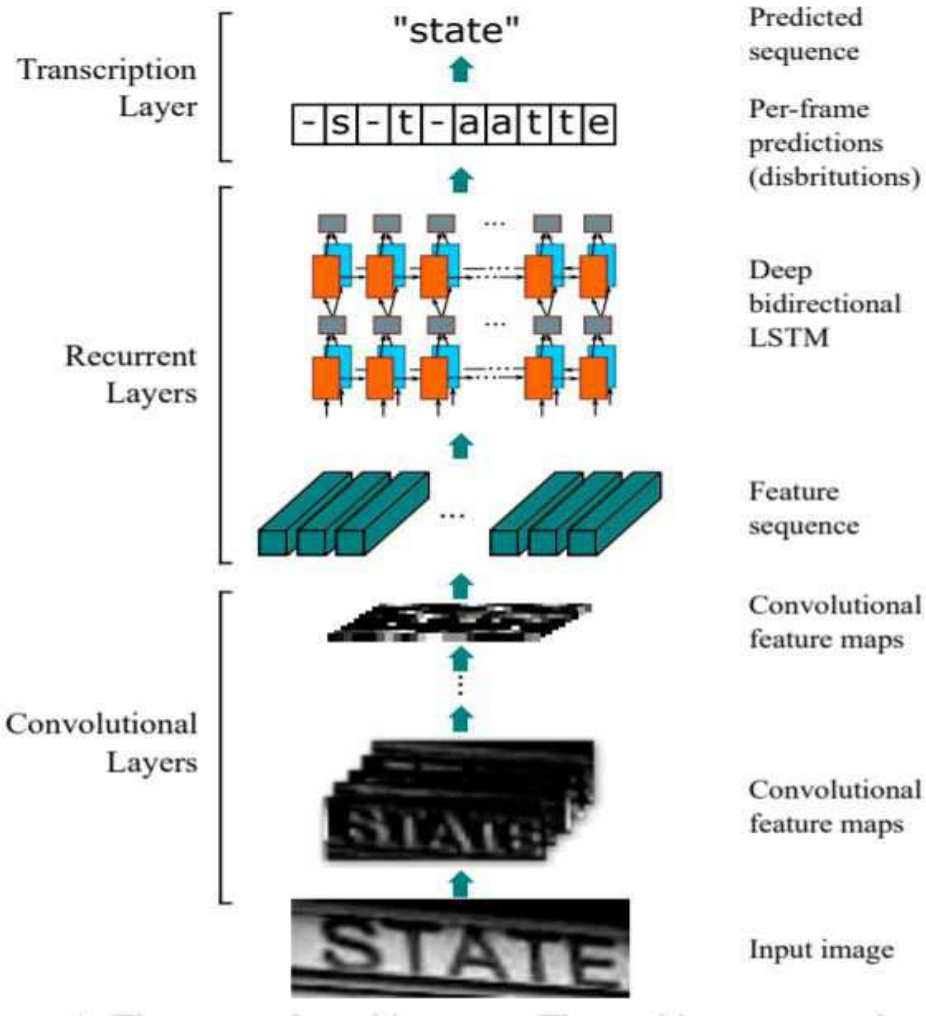

Рисунок 9 – Визуализация архитектуры CRNN

В статье [18] авторы предлагают использовать нейронную сеть без рекуррентных слоев для улучшения параллелизации вычислений. Входное изображение обрабатывается скользящим окном, далее сверточными слоями извлекаются признаки и передаются специальному сверточному слою с вниманием (attention) для кодирования и декодирования последовательностей. Данная архитектура представлена на рисунке 12.

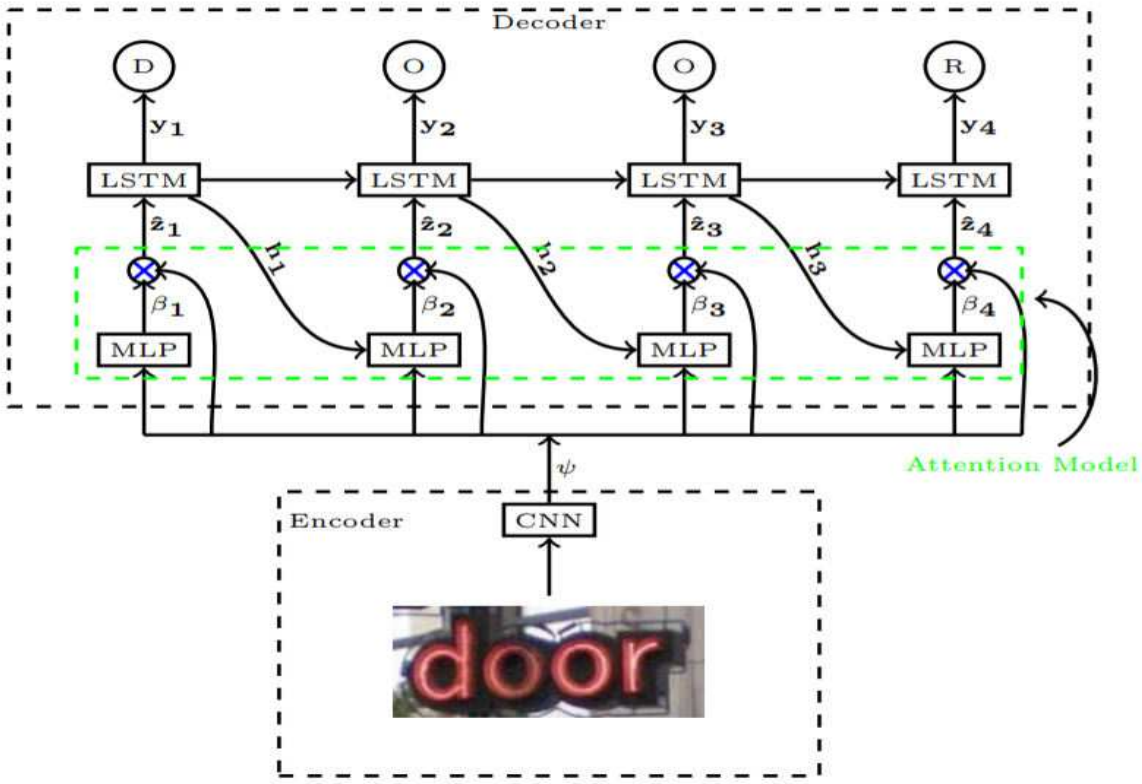

Рисунок 10 – Архитектура нейронной сети с вниманием (attention)

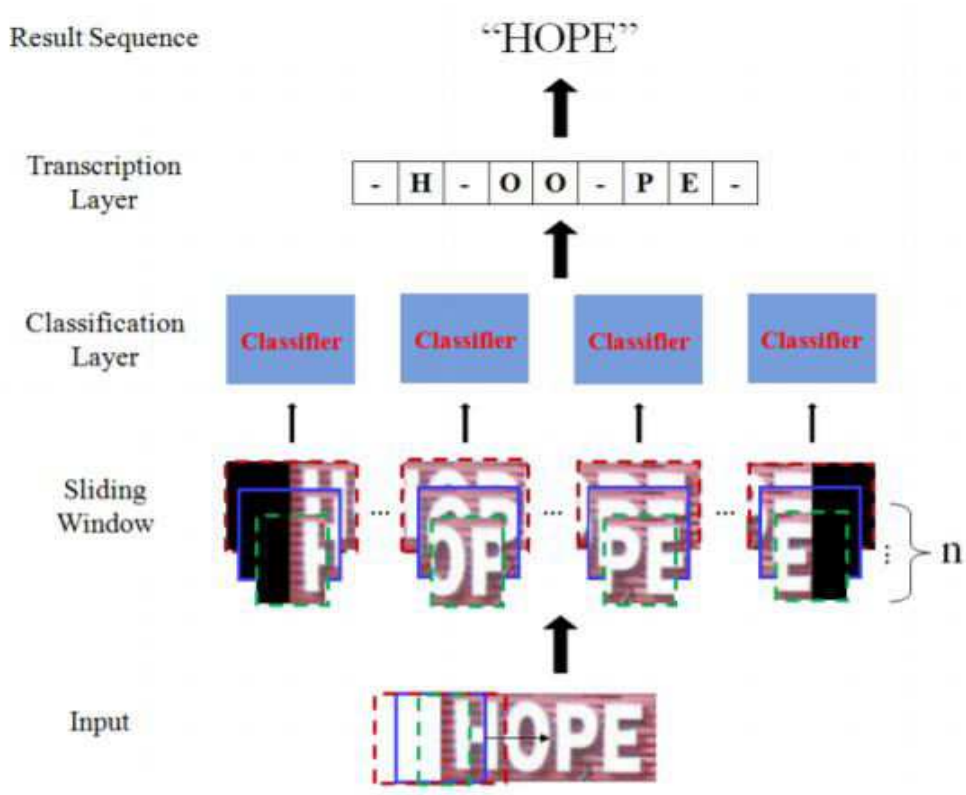

Рисунок 11 – Архитектура нейронной сети, предлагаемой авторами [17]

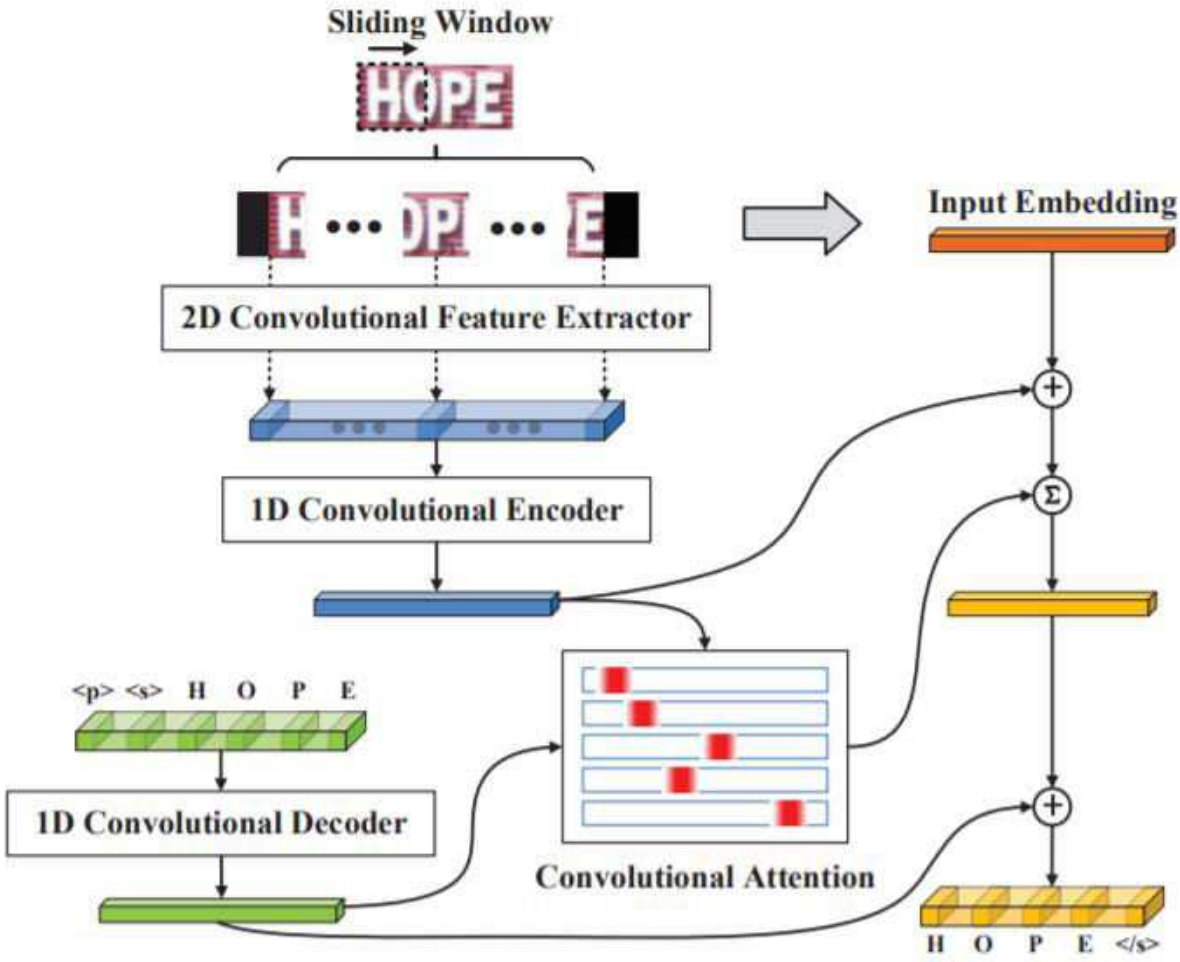

Рисунок 12 – Архитектура нейронной сети, предлагаемой в статье [18]

В работе [19] предлагается использовать сверточные нейронные сети без рекуррентных слоев с 2D-Attention. Визуализация вычислительного графа данной сети представлена на рисунке 13.

Авторы работы [20] предлагают масштабировать входное изображение по ширине n раз, и подавать все n изображений на вход сверточной сети с общими весами для всех изображений (weight sharing). Далее извлеченные сверточной сетью признаки обрабатываются механизмом внимания (attention). Архитектура сети представлена на рисунке 14.

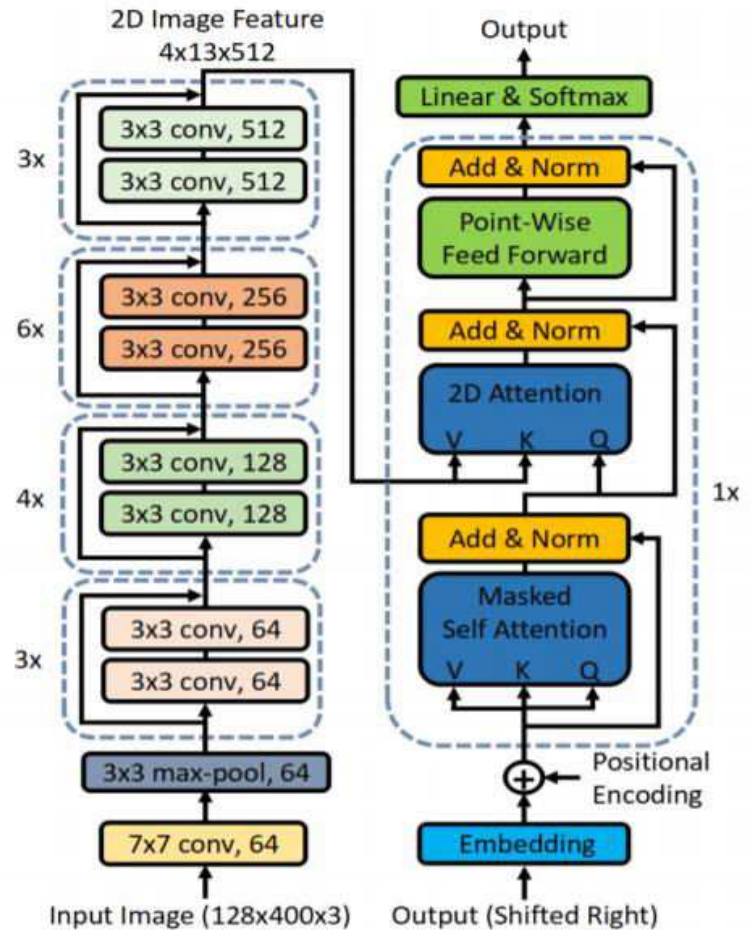

Рисунок 13 – Визуализация вычислительного графа нейронной сети, предложенной в [19]

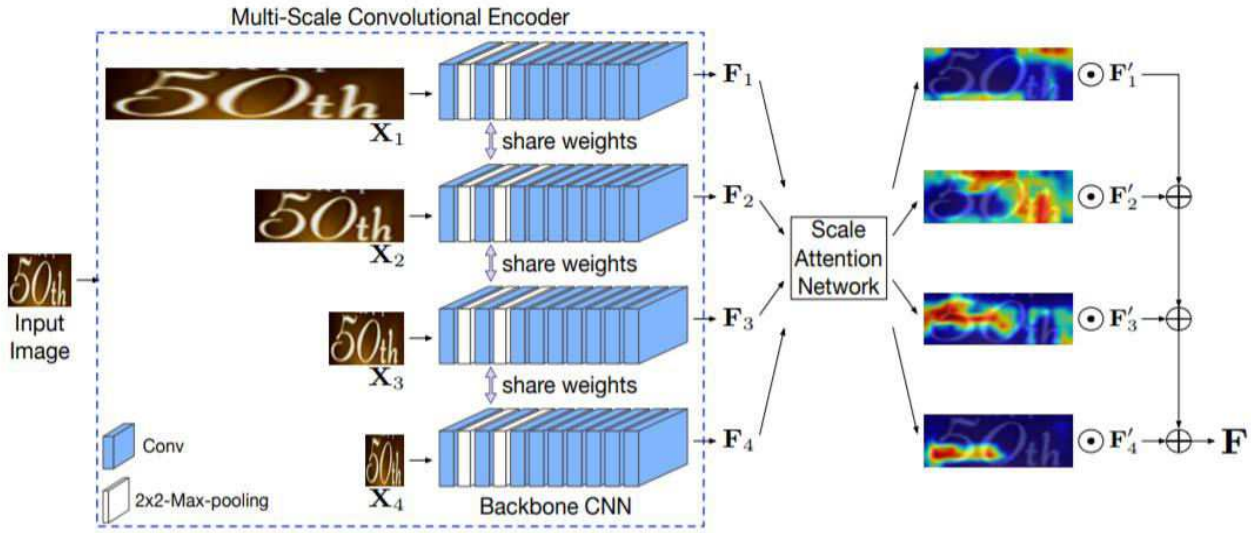

Рисунок 14 – Архитектура сети с различным масштабированием входного изображения по ширине

Во многих из обозреваемых подходов после обучения нейронной сети декодированное слово проверяется по заранее заданному ограниченному словарю. Это часто позволяет добиться большей точности распознавания, но делает систему неспособной к распознаванию слов, отсутствующих в словаре.

2 Подход с применением CRNN-архитектуры нейронных сетей

Для решения задачи классификации локализованного текста на сложной графической сцене предлагается использовать сочетание сверточной нейронной сети (CNN), рекуррентной нейронной сети (RNN) и специализированной Connectionist Temporal Classification функции ошибки (CTC-loss) [14, 15].

Ниже описан принцип работы всех слоев, используемых в данной архитектуре сети.

2.1 Полносвязный слой

Вычисление прямого распространения полносвязного слоя нейронной сети – это линейное преобразование вектора входных данных $x=(x_1, x_2, \ldots, x_m)$ с матрицей весов W размерности n x (m+1), где каждый элемент преобразованного вектора впоследствии передается нелинейной функции активации f:

$$f(Wx) = \begin{pmatrix} f(w_1^T x) \\ \vdots \\ f(w_n^T x) \end{pmatrix}, \quad (1)$$

где x – вектор входных данных;

b – вектор смещения (сдвига) слоя;

n – количество нейронов в слое;

f – функция активации слоя;

W - матрица весов слоя.

На практике часто вместо прибавления к результату линейного преобразования отдельного вектора смещения нейронов (bias) размерности n матрица весов W конкатенируется со столбцом из единиц, что математически равнозначно.

$$W = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} w_{11} & \dots & w_{1m} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ w_{n1} & \dots & w_{nm} & 1 \end{pmatrix}, \quad (2)$$

где n – количество нейронов в слое;

m – размерность входных данных.

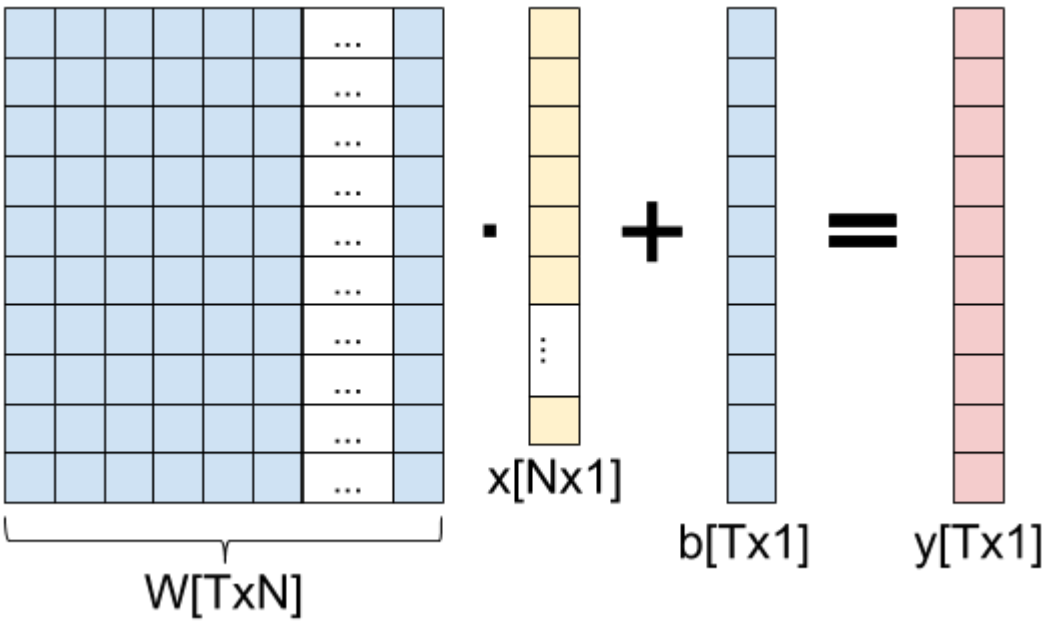

Рисунок 15 – Визуализация вычисления полносвязного слоя

Обратное распространение ошибки по полносвязному слою представляет собой градиент композиции функций:

$$\frac{\partial t}{\partial x} = \frac{\partial f}{\partial x} * \frac{\partial t}{\partial f}; \quad (3)$$

$$\frac{\partial t}{\partial W} = \frac{\partial f}{\partial W} * \frac{\partial t}{\partial f}, \quad (4)$$

где t – значения нейронов последующего слоя;

f – функция активации данного слоя;

x – значения нейронов данного слоя;

W – матрица весов данного слоя.

Градиент по слою x – это ошибка, которая в последствии передается на предыдущий слой сети, а градиент по матрице весов W используется для изменения значений весов при помощи оптимизационного алгоритма.

2.2 Сверточный слой

Сверточный слой нейронной сети представляет собой взвешенную с помощью матрицы весов W сумму значений предыдущего слоя. Математически прямое распространение сверточного слоя с матрицей весов W размерности (2d+1) x (2d+1) для элемента i-й строки и j-го столбца входной матрицы x описывается так:

$$y_{i,j} = \sum_{-d \le a,b \le d} W_{a,b} x_{i+a,j+b}, \quad (5)$$

где $y_{i,j}$ – результат свертки для элемента с координатами (i, j);

$x_{i+a,j+b}$ – элемент входной матрицы с координатами (i+a, j+b)

$W_{a,b}$ – элемент матрицы весов с координатами (a, b).

Принцип работы сверточного слоя представлен на рисунке 16. Размер матрицы последующего слоя рассчитывается по следующей формуле:

$$(w, h) = (mW - kW + 1,\ mH - kH + 1), \quad (6)$$

где w - ширина матрицы последующего слоя;

h - высота матрицы последующего слоя;

mW - ширина матрицы предыдущего слоя;

mH - высота матрицы предыдущего слоя;

kW - ширина ядра свертки;

kH - высота ядра свертки.

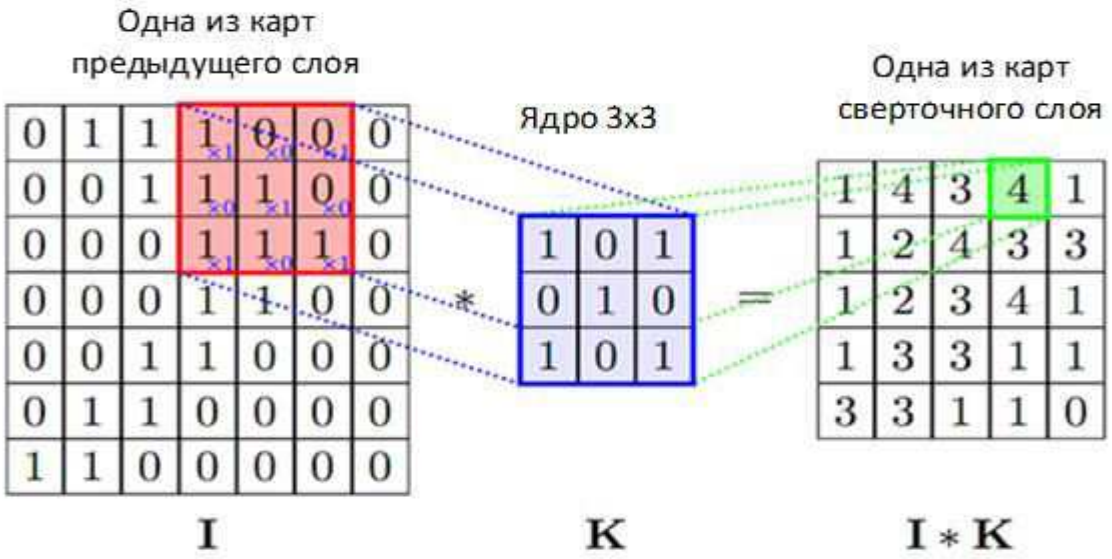

Рисунок 16 - Принцип работы сверточного слоя

Часто для достижения той же размерности, что и на входе сверточного слоя используется т.н. padding – заполнение краев исходной матрицы нулями до требуемой размерности. Как правило, на одном сверточном слое вычисляется сразу несколько карт признаков, каждая с собственными весами относительно каждой входной карты признаков.

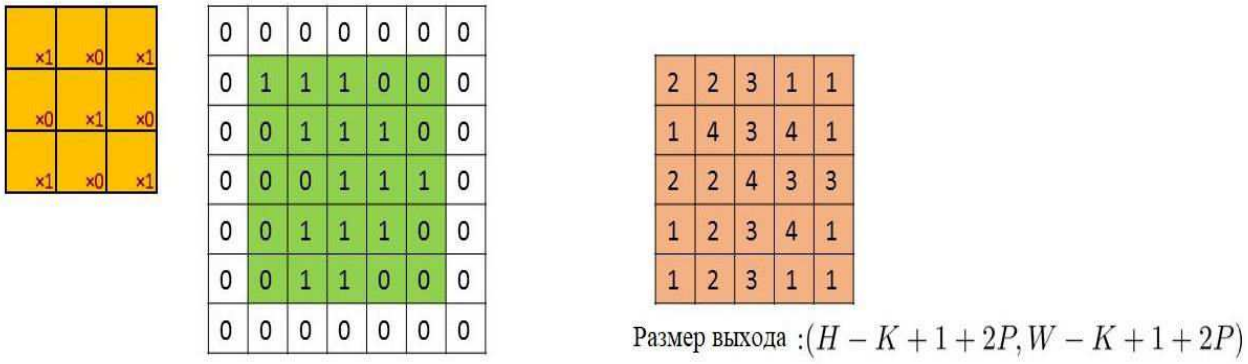

Рисунок 17 – Визуализация принципа работы операции свертки с padding

2.3 Слой субдискретизации

Слой субдискретизации (подвыборочный слой, pooling-слой), как правило, располагается после сверточного слоя, переданного нелинейной функции активации. Назначение данного слоя – это понижение размерности карт признаков с целью экономии вычислительных мощностей при обучении сети и регуляризации. Наиболее распространенный вариант субдискретизации – это max-pooling, применение операции взятия максимума для плавающего окна с заранее заданным шагом и размером.

Принцип работы слоя субдискретизации с окном размера 2 x 2 и шагом 2 представлен на рисунке 18:

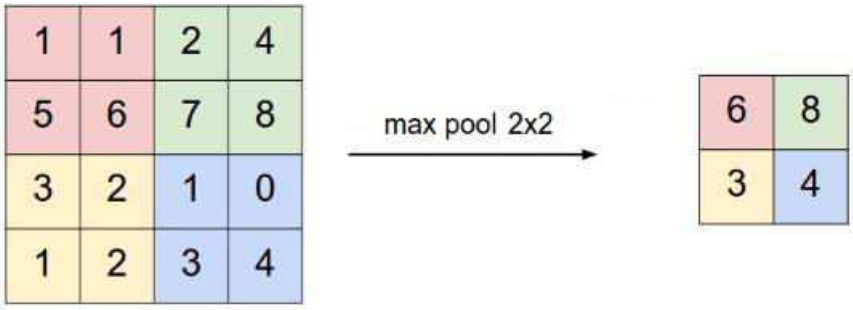

Рисунок 18 – Визуализация принципа работы слоя субдискретизации

Операция max-pooling’а описывается следующей формулой:

$$y_{i,j} = max_{-d \le a \le d, -d \le b \le d}(x_{i+a,j+b}), \quad (7)$$

где x - выход предыдущего слоя;

d – размер окна субдискретизации;

$y_{i,j}$ – результат применения max-pooling’а для элемента с координатами (i,j).

2.4 Слой нормализации по мини-батчам

Предложенный в статье [21] метод нормализации по мини-батчам используется сейчас в большинстве архитектур нейронных сетей. Сети с использованием нормализации по мини-батчам значительно более устойчивы к инициализации весов и менее склонны к переобучению. Для каждого из нейронов вычисляется математическое ожидание μ_x и стандартное отклонение σ_x по мини-батчу $x=\{x_1, x_2, \ldots, x_m\}$.

$$\mu_x = \frac{1}{m}\sum_{i=1}^{m} x_i\,; \tag{8}$$

$$\sigma_x^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_x)^2, \tag{9}$$

где m – размер мини-батча;

x – множество значений нейрона по всему мини-батчу;

Затем данные нормализуются посредством вычитания математического ожидания и деления на стандартное отклонение.

$$x_i' = \frac{x_i - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}}; \tag{10}$$

где ε – дополнительная константа, вводимая для предотвращения деления на ноль;

Вычисляется конечный результат с учетом дополнительного масштабирования и сдвига нормализации.

$$y_i = \gamma x_i + \beta, \tag{11}$$

где γ – дополнительный оптимизируемый параметр для масштабирования нормализации по каждому элементу;

β – дополнительный оптимизируемый параметр для сдвига нормализации по каждому элементу.

Также нейронные сети, в которых используется нормализация по мини-батчам сходятся на большинстве задач быстрее, чем сети без неё. Авторы оригинальной статьи по CRNN [15] используют 2 слоя батч-нормализации в своей архитектуре.

2.5 Рекуррентный слой

В архитектуре CRNN признаки, извлекаемые сверточными слоями, являются последовательностями, поскольку в распознаваемом тексте важен порядок следования символов друг за другом. Их последовательный характер должен учитываться при обучении сети, и наиболее распространенный способ учитывать его – использовать рекуррентные слои.

По характеру последовательностей задачи машинного обучения условно делятся на 5 типов:

- один к одному (one to one) – один вход модели соответствует одному выходу (например, задачи классификации образов);

- один ко многим (one to many) – один вход модели соответствует последовательности выходов (например, задачи аннотирования изображений);

- многие к одному (many to many) – последовательность входов соответствует одному выходу (например, задачи анализа тональности текста);

- многие ко многим (many to many) – последовательность входов соответствует последовательности выходов (например, машинный перевод);

- многие ко многим с синхронизацией (many to many) – синхронизированная последовательность входов и выходов (например, классификация видео с маркировкой каждого кадра).

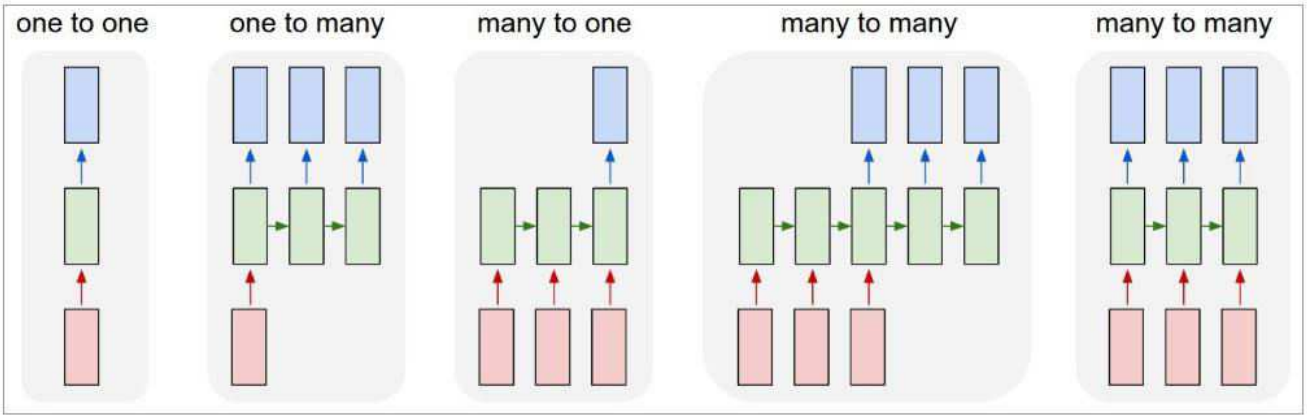

Рисунок 19 – Условное деление задач машинного обучения по характеру последовательностей

Решаемая задача относится к 4-му типу, в котором последовательность входов соответствует последовательности выходов без взаимной синхронизации. Последовательность признаков, извлекаемых сверточными слоями, соответствует переменной длины последовательности выходных символов в распознаваемом слове.

Основной принцип работы рекуррентных слоев – это использование значений нейронов, полученных на предыдущих итерациях обучения сети. В слое задается некоторое скрытое состояние s, которое сохраняется и передается от предыдущего к следующему элементу последовательности.

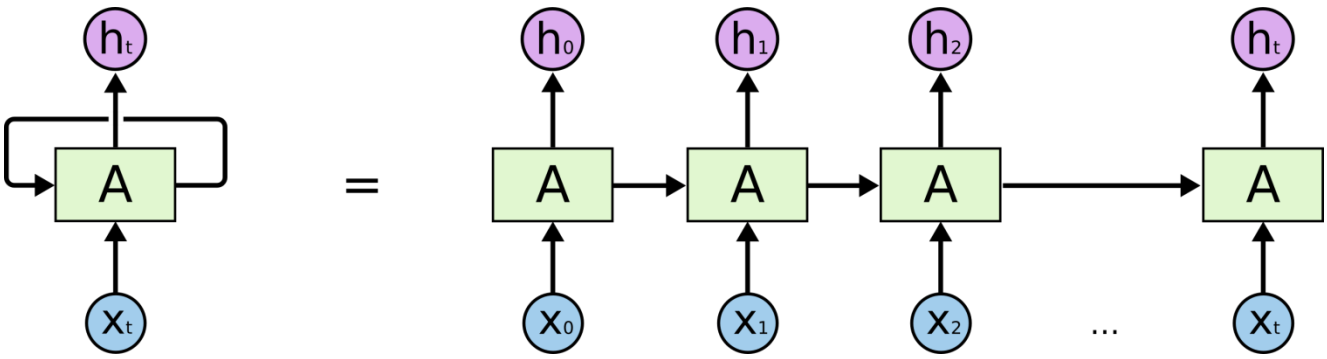

Рисунок 20 – Принцип работы рекуррентного слоя и его развернутое во времени представление

Математически классический рекуррентный слой (Vanilla RNN) нейронной сети описывается следующим образом.

$$s_t = f(Ws_{t-1} + Ux_t + b); \quad (12)$$

$$h_t = g(Vs_t + c), \quad (13)$$

где x_t – входные данные на текущем временном шаге;

s_{t-1} – значения скрытого состояния на предыдущем временном шаге;

W – матрица весов для скрытого состояния;

U – матрица весов для входных данных;

b – смещение данных перед нелинейностью;

f – нелинейная функция активации рекуррентного слоя (обычно используется гиперболический тангенс, tanh);

s_t – значение скрытого состояния на текущем временном шаге;

V – матрица весов на выходе слоя;

c – смещение данных на выходе слоя;

g – функция нелинейности на выходе (например, softmax);

h_t – выходные значения слоя.

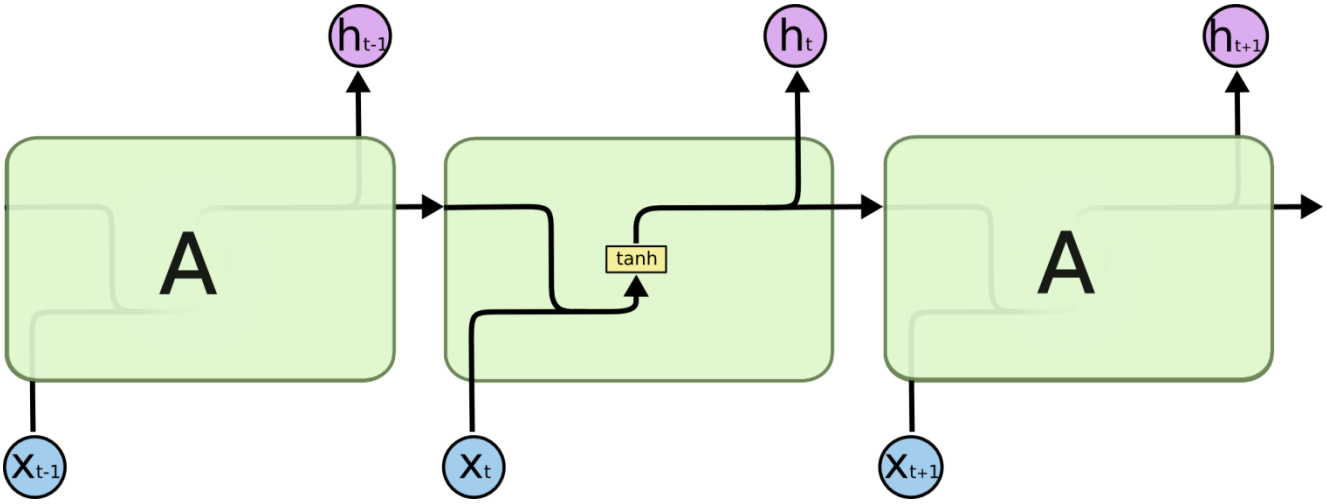

Рисунок 21 – Структура классического рекуррентного слоя

Но у классической архитектуры рекуррентных сетей есть несколько проблем. Одна из них – проблема длинных зависимостей. При матричном перемножении информация о долгосрочных зависимостях на более ранних временных шагах будет постоянно перезаписываться, и сеть таким образом будет иметь более “короткую” память и неспособна воспроизводить долгосрочные зависимости. Эта проблема частично решается применением более сложных архитектур, одна из них – LSTM (Long Short-Term Memory). В сети LSTM есть 3 т.н. узла или гейта (gate): забывающий гейт (forget gate), входной гейт (input gate), выходной гейт (output gate) и сама рекуррентная ячейка со скрытым состоянием (cell state).

Забывающий гейт f_t необходим для контроля данных, поступающих с предыдущих временных шагов. Он описывается следующей формулой:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{14}$$

где h_{t-1} – скрытое состояние на предыдущем временном шаге;

x_t – входные данные на текущем временном шаге;

W_f – матрица весов забывающего гейта;

b_f – смещение забывающего гейта;

σ – нелинейная функция активации (сигмоида).

Входной гейт i_t обрабатывает входные данные для текущего временного шага и описывается формулой:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{15}$$

где W_i – матрица весов входного гейта;

b_i – смещение входного гейта;

После вычисления значений забывающего и входного гейта вычисляется значение-кандидат на изменение состояния ячейки C_t^I (candidate cell gate) и состояние ячейки C_t (cell state) LSTM:

$$C_t' = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C); \quad (16)$$

$$C_t = f_t * C_{t-1} + i_t * C_t', \quad (17)$$

где W_C – матрица весов состояния ячейки;

b_C – смещение состояния ячейки;

tanh – нелинейная функция активации (гиперболический тангенс);

В выходном гейте вычисляется скрытое состояние в текущий момент времени h_t и значение выходного гейта o_t.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o); \quad (18)$$

$$h_t = o_t * \tanh(C_t), \quad (19)$$

где W_o – матрица весов выходного гейта;

b_o – смещение выходного гейта;

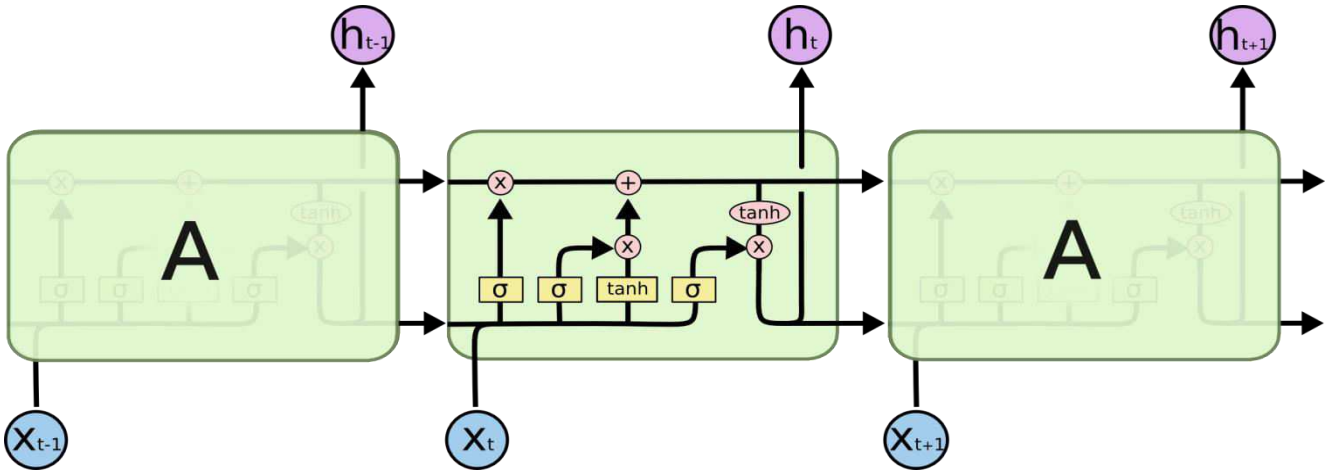

Рисунок 22 – Структура рекуррентного LSTM (long short-term memory) слоя

Поскольку при подаче изображения на вход сети известно как начало последовательности, так и её конец – используются двунаправленные (bidirectional) LSTM слои. Принцип работы bidirectional LSTM таков – задаются 2 независимых LSTM слоя, в одном из которых скрытое состояние передается от начала последовательности к её концу, а в другом наоборот – от конца последовательности к её началу, после чего эти 2 слоя конкатенируются друг с другом. Это позволяет сети в процессе обучения учитывать для каждого символа не только информацию о предыдущих символах, но и информацию о последующих.

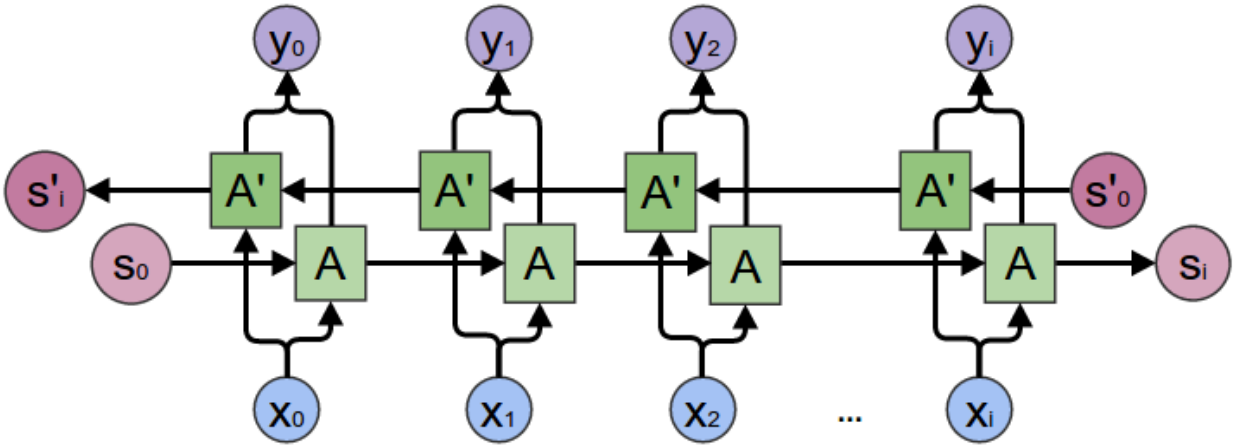

Рисунок 23 – Принцип работы двунаправленного (bidirectional) LSTM слоя

2.6 CTC loss

Прежде, чем описать непосредственно CTC (Connectionist temporal classification) loss, рассмотрим полностью архитектуру CRNN.

Входное изображение подается на вход слоям сверточной нейронной сети[22, 34] для последующего извлечения признаков.

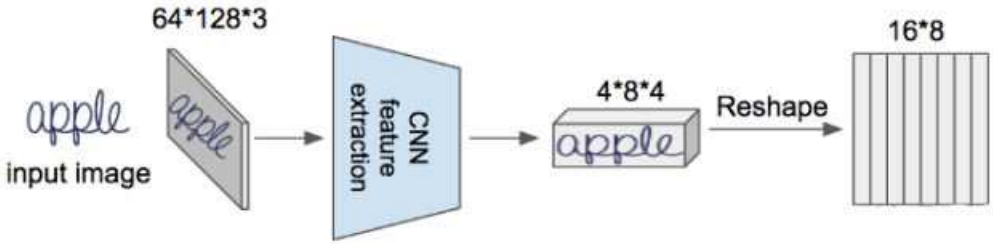

Рисунок 24 – Пример извлечения признаков при помощи сверточной нейронной сети

Выход сверточной сети представляет собой трехмерный массив, который необходимо преобразовать к двухмерному, вторая размерность которого должна быть равна максимальному количеству символов во входном изображении.

После преобразования каждый столбец матрицы подается на вход соответствующей LSTM-клетке [23]. Обработка извлеченных признаков рекуррентным слоем визуализирована на рисунке 25.

Рекуррентный слой необходим для представления входных признаков в виде последовательности и распознавания сетью символов именно в том порядке, в котором они находятся в искомом тексте.

После обработки рекуррентным слоем каждый вектор передается полносвязному слою и активационной функции softmax [22]. Размерность полносвязного слоя равна длине массива всех возможных распознаваемых символов с добавлением blank-маркера.

Данный маркер алгоритм CTC-ошибки использует для обозначения отсутствия символа в тексте, т.к. большая часть всех подаваемых на вход последовательностей символов будет по длине меньше максимально возможной. Softmax-функция же представляет входной вектор в виде вектора распределения вероятностей для всех возможных символов, включая и маркер отсутствия символа(blank) [15]. Пример работы декодирующего алгоритма приведен на рисунке 26.

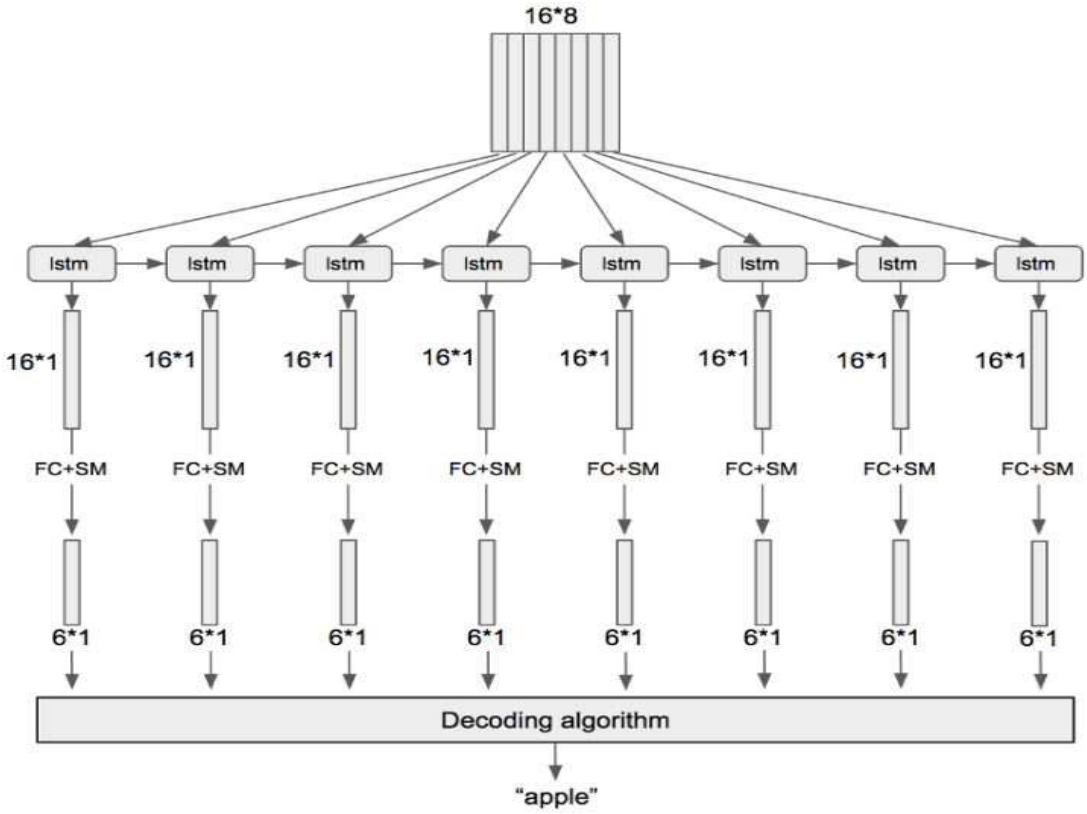

Рисунок 25 – Обработка извлеченных признаков рекуррентным слоем

Выход представляет собой матрицу с распределенными функцией softmax по столбцам вероятностями, где количество столбцов – максимально возможная длина слова, а количество строк – длина "алфавита" L + 1(blank-маркер). Данная матрица подается на вход декодирующему алгоритму, который сперва удаляет все повторяющиеся символы, а затем blank-маркеры. На рисунке ниже приведен пример декодирования входного пути.

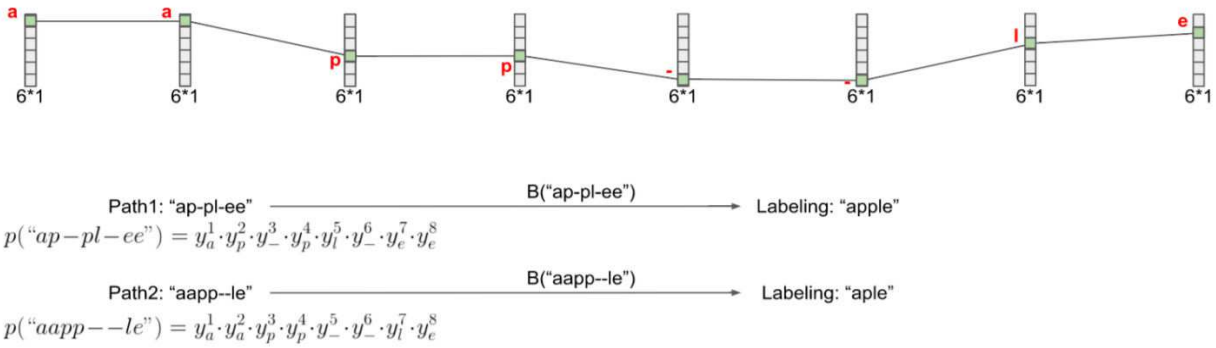

Рисунок 27 – Визуализация работы декодирующего алгоритма

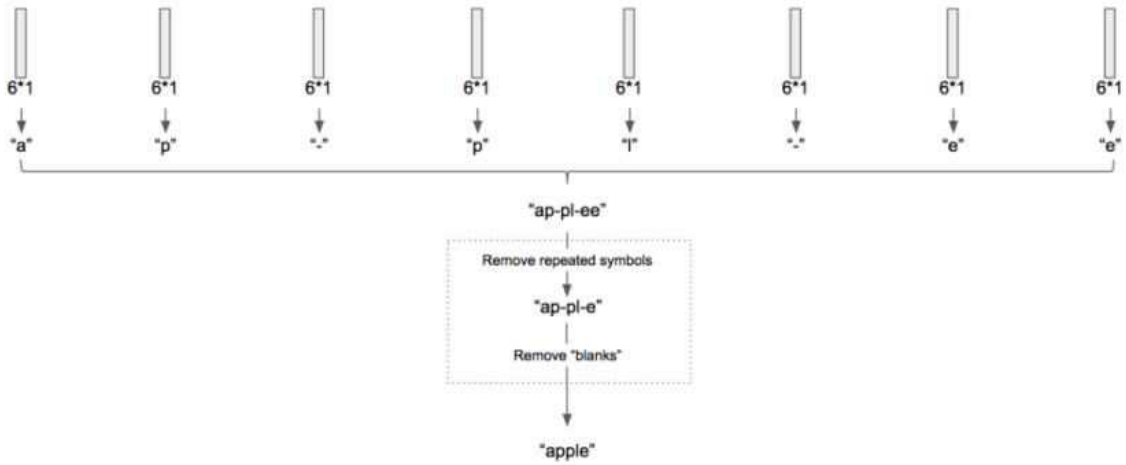

Рисунок 26 – Пример работы декодирующего алгоритма

Итоговый алфавит – это множество всех возможных символов в слове и blank-маркер:

$$L' = L \cup \{\text{blank}\}; \tag{20}$$

Вероятность пути Y равна произведению вероятностей всех активированных ячеек каждого столбца матрицы:

$$p(Y) = \prod_{t=1}^{T} y_{Y_t}^{t}, \quad \forall Y \in L'; \tag{21}$$

Пример расчета вероятности пути “ap-pl-ee”:

$$p(\text{"ap-pl-ee"}) = y_a^1 * y_p^2 * y_-^3 * y_p^4 * y_l^5 * y_-^6 * y_e^7 * y_e^8; \tag{22}$$

Одному слову всегда соответствует множество путей, и для расчета вероятности слова необходимо рассчитать сумму вероятностей по всем возможным путям [14]. Сама функция ошибки напоминает бинарную кросс-

энтропию, только в качестве вероятности класса здесь используется вероятность слова.

$$\text{CTC loss} = -\ln(\text{p("apple")}); \tag{23}$$

Для случая из примера (в алфавите 6 символов, максимальное количество символов – 8) это 6^8=1679616 возможных путей. На практике же длина алфавита и максимальное число символов порой в разы больше, поэтому для нахождения вероятности слова намного эффективнее использовать динамическое программирование.

Создается новая матрица, подобная полученной на выходе softmax-слоя.

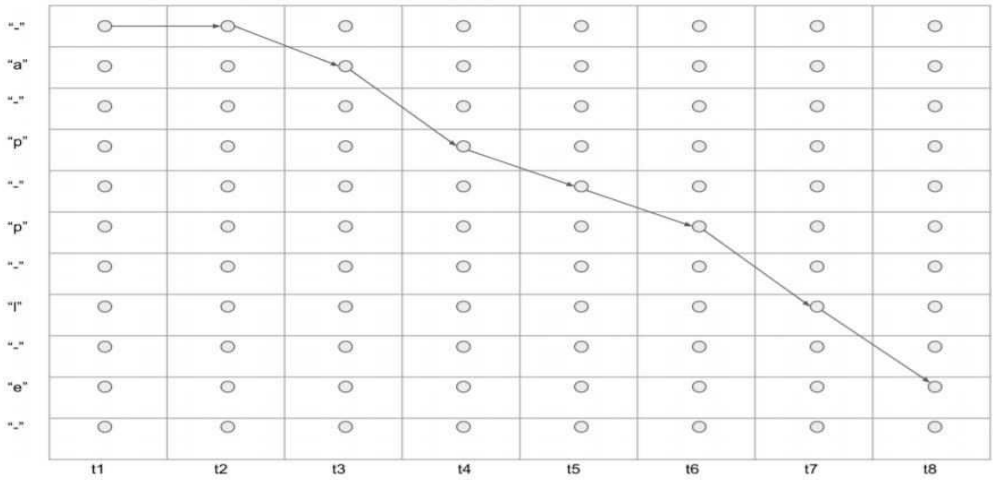

Рисунок 28 – Новая матрица для построения множества всех возможных путей

С её помощью можно построить множество всех возможных путей для нужного слова, а так же рассчитать коэффициент α. Расчет коэффициента α имеет вид:

$$\alpha_t(s) = \left(\alpha_{t-1}(s) + \alpha_{t-1}(s-1)\right) * y^t_{seq(s)}; \tag{24}$$

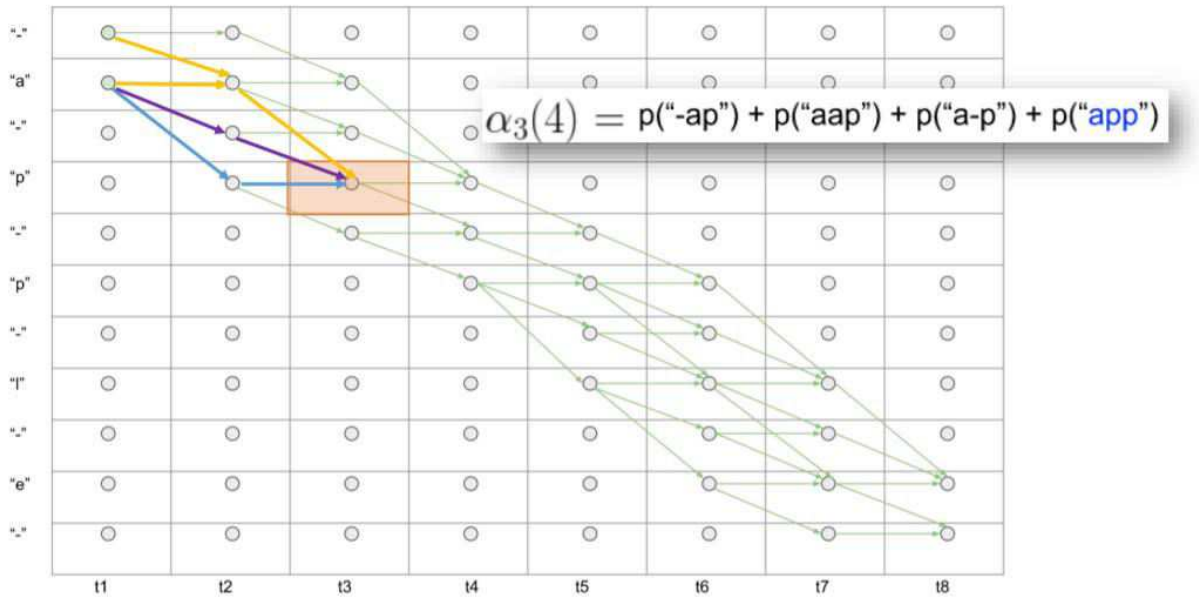

Рисунок 29 – Визуализация всех возможных путей для слова “apple”

Для каждого элемента t-го столбца и s-й строки матрицы рассчитывается сумма всех путей, ведущих к данному элементу, после чего умножается на вероятность самого элемента. Данная операция повторяется до последнего столбца, значения которого суммируются и передаются функции ошибки.

Данный подход позволяет эффективно вычислить сумму всех возможных путей до требуемого слова и получить его вероятность, что необходимо для обучения нейронной сети.

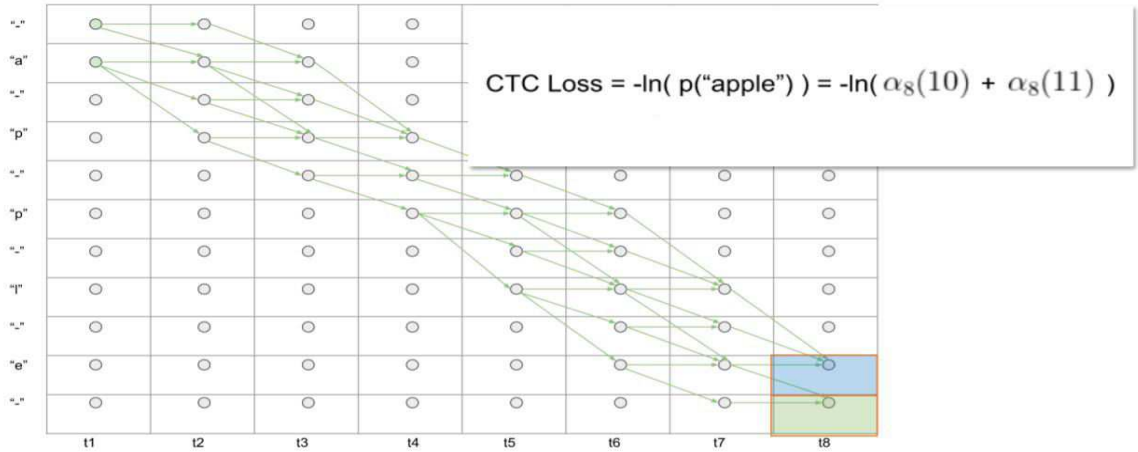

Рисунок 30 – Расчет суммы всех путей с применением динамического программирования

Коэффициент α вводится для прямого распространения, для обратного используется другой коэффициент – β.

$$\beta_t(s) = \left(\beta_{t+1}(s) + \beta_{t+1}(s+1)\right) * y^t_{seq(s)}; \quad (25)$$

Коэффициент β для элемента матрицы рассчитывается для всех путей, выходящих из данного элемента. Так, поделив произведение α_s^t и β_s^t на вероятность элемента y_s^t и просуммировав вдоль t-го столбца матрицы, получим сумму всех возможных путей:

$$p("apple") = \sum_{s=1}^{seq} \frac{\alpha_s^t * \beta_s^t}{y_s^t}; \quad (26)$$

Для обучения сети градиентным спуском необходимо продифференцировать функцию ошибки по всем выходам сети y_k^t:

$$\frac{\partial(-\ln(p("apple")))}{\partial y_k^t} = -\frac{1}{p("apple")} * \frac{\partial p("apple")}{\partial y_k^t}; \quad (27)$$

Рассматриваются только пути, идущие через символ k в момент t:

$$\frac{\partial p("apple")}{\partial y_k^t} = -\frac{1}{{y_k^t}^2} * \sum_{seq(s)=k} \alpha_t(s) * \beta_t(s); \quad (28)$$

Таким образом, архитектура CRNN является полностью дифференцируемой системой (end-to-end) и представляет собой один вычислительный граф.

Для улучшения точности распознавания декодированное слово часто проверяют по заданному словарю. В данной работе для этого между декодированным словом и каждым словом из словаря вычисляется расстояние Левенштейна, после чего выбирается пара слов с наименьшим расстоянием.

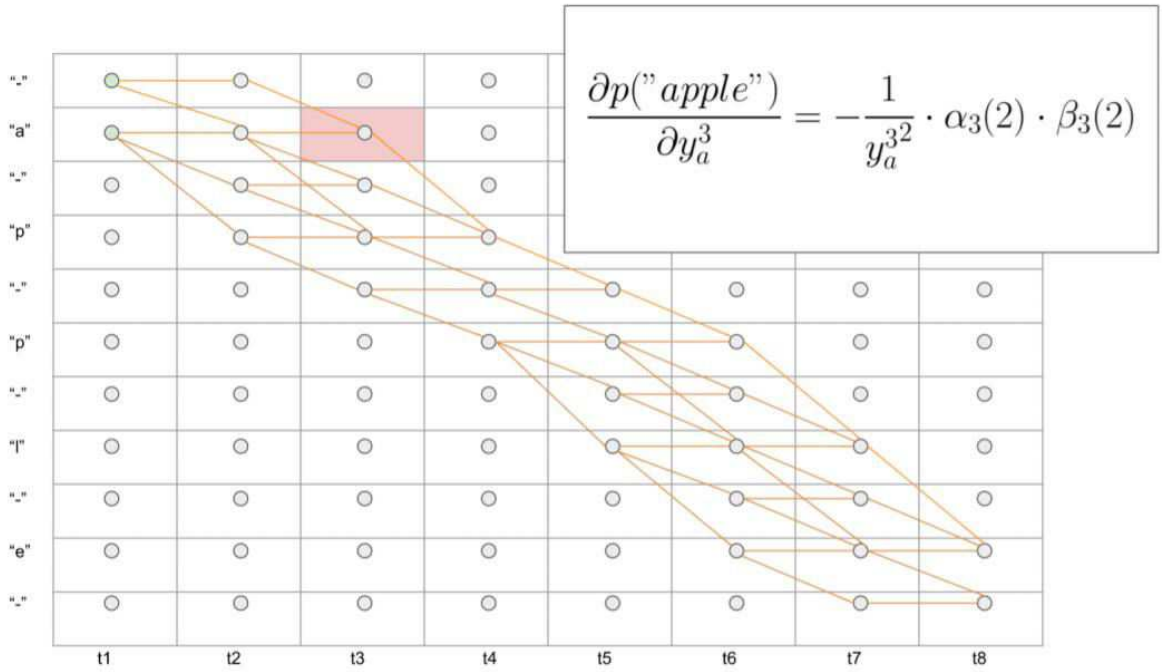

Рисунок 31 – Пример дифференцирования по символу «a» в момент «3»

3 Эксперимент

Поскольку из существующих на данный момент наборов данных для данной задачи нет наборов с количеством изображений, достаточным для качественного обучения сети – было принято решение использовать набор Synth 90k, предложенный и разработанный в статье [2]. Набор данных представляет собой программно сгенерированные изображения в градациях серого переменной ширины и высотой в 31 пиксель. В тренировочной части набора содержится более 7 миллионов изображений с 90000 уникальными английскими словами. Примеры изображений представлены на рисунке 32.

Рисунок 32 - Примеры изображений из набора данных Synth 90k

Для оценки качества натренированной сети использовались два набора данных. Первый набор – IIIT 5K. Данный набор представляет собой локализованный текст на фотографиях магазинных витрин, рекламных щитов, объявлений, уличных знаков и т.п. В наборе содержится 2000 тренировочных и 3000 тестовых изображений. Примеры изображений представлены на рисунке 33.

Рисунок 33 - Примеры изображений из набора данных IIIT5K

Второй набор данных – SVT (Street View Text). В наборе данных, по аналогии с IIIT 5K-Word, преобладают изображения с текстом на магазинных

витринах, рекламных щитах, вывесках различных заведений. Набор данных содержит 249 изображений графических сцен и 647 изображений локализованного текста. Под тренировочную выборку выделено 80% изображений, под тестовую – 20%. Примеры изображений представлены на рисунке 34.

Рисунок 34 - Примеры изображений из набора данных SVT (Street View Text)

Была разработана программа на ЭВМ с реализацией CRNN+CTC-loss архитектуры на языке программирования Python с использованием фреймворка Tensorflow. Входное изображение приводилось к размеру 31x100 пикселей в градациях серого, после чего подавалось на вход нейронной сети вместе с соответствующим словом. Разработанная нейронная сеть имеет следующую архитектуру:

- 1-й сверточный слой, функция активации ReLU, ядро 3x3, (31, 100, 64);
- 1-й max-pooling слой, ядро 2x2, шаг 2x2, (16, 50, 64);
- 2-й сверточный слой, функция активации ReLU, ядро 3x3, (16, 50, 128);
- 1-й слой батч-нормализации;
- 2-й max-pooling слой, ядро 2x2, шаг 2x2, (8, 25, 128);
- 3-й сверточный слой, функция активации ReLU, ядро 3x3, (8, 25, 256);
- 2-й слой батч-нормализации;
- 4-й сверточный слой, функция активации ReLU, ядро 3x3, (8, 25, 256);
- 3-й max-pooling слой, ядро 2x2, шаг 1x2, (8, 13, 256);
- 5-й сверточный слой, функция активации ReLU, ядро 3x3, (8, 13, 512);
- 3-й слой батч-нормализации;
- 6-й сверточный слой, функция активации ReLU, ядро 3x3, (8, 13, 512);

- 4-й max-pooling слой, ядро 2x2, шаг 1x2, (8, 7, 512);

- 7-й сверточный слой, функция активации ReLU, ядро 3x3, (8, 7, 512);

- 1-й bidirectional-LSTM слой, (256, 256);

- 2-й bidirectional-LSTM слой, (256, 256);

- 1-й полносвязный слой, функция активации Softmax;

- CTC-loss.

В качестве оптимизатора выбран Adam с шагом обучения 0.001, размер входного пакета (batch), для которого происходит единовременная корректировка весов – 64. Длина алфавита – 43 символа, максимально возможная длина слова – 16 символов. Веса сети до обучения инициализируются нормально распределенными значениями с математическим ожиданием, равным 0 и стандартным отклонением, равным 0.1. Для крайних пикселей в сверточных слоях используется автоматическое заполнение значений нулями для получения на выходе слоя той же размерности, что и на входе. Нейронная сеть обучалась 1.5 эпохи (7 224 612 изображений из Synth 90k) на графическом процессоре NVIDIA GeForce GTX 650, процесс обучения занял 170 часов (7 дней).

График изменения ошибки на тренировочных данных (Synth 90k), SVT и IIIT5K приведен на рисунке 35.

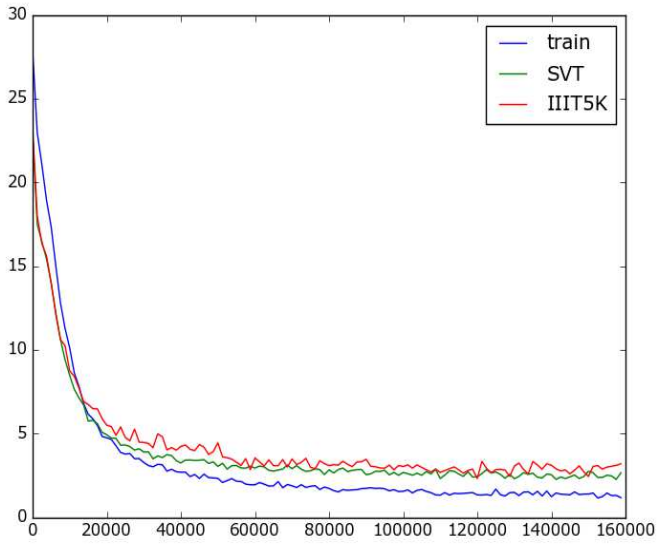

Рисунок 35 - График изменения ошибки нейронной сети

График изменения расстояния Левенштейна между предсказанными и эталонными словами приведен на рисунке 36.

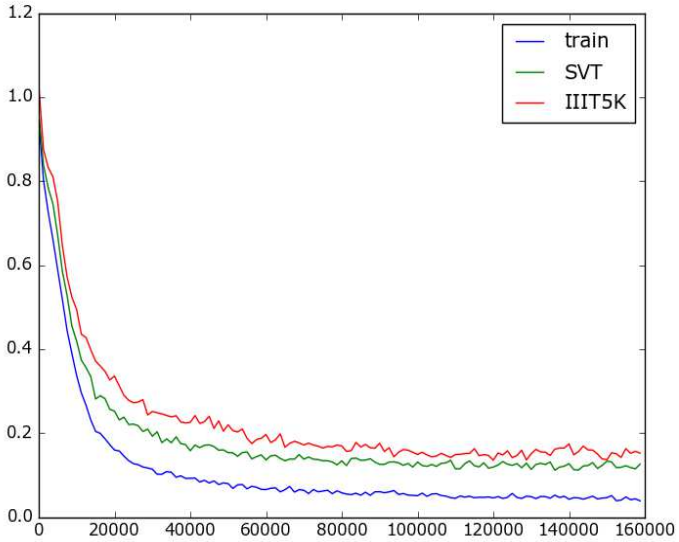

Рисунок 36 - График изменения расстояния Левенштейна

График изменения точности (отношения правильно предсказанных слов к общему числу) приведен на рисунке 37.

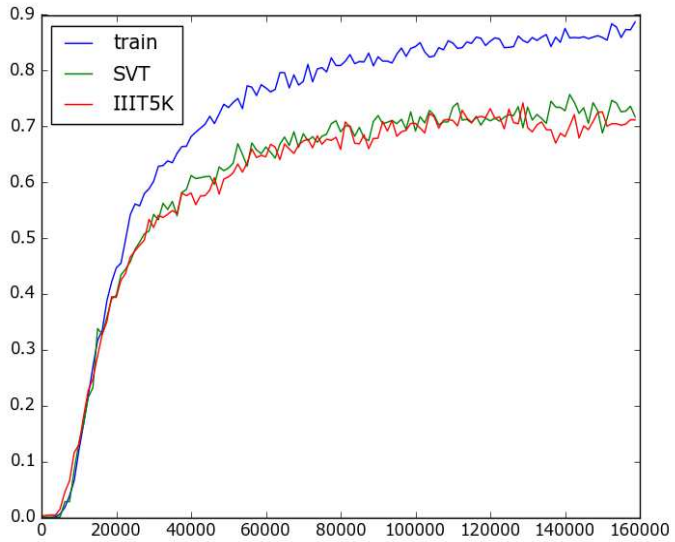

Рисунок 37 - График изменения точности

В таблице 1 приведены результаты эксперимента и сравнение с результатами других подходов. Система показывает результаты, близкие к state-of-the-art решениям, при этом обучаясь быстрее CRNN из [4] благодаря дополнительным слоям батч-нормализации.

Таблица 1 – Результаты эксперимента и сравнение итоговой точности с другими подходами на двух наборах данных

Метод	IIIT5K			SVT	
	50 слов	1000 слов	без словаря	50 слов	без словаря
ABBYY	24.3	-	-	35.0	-
Wang и др.	-	-	-	57.0	-
Mishra и др.	64.1	57.5	-	73.2	-
Wang и др.	-	-	-	70.0	-
Rodrguez-Serrano и др.	76.1	57.4	-	70.0	-
Bissacco и др. [5]	-	-	-	-	78.0
Jaderberg и др.	-	-	-	86.1	-
Jaderberg и др. [2]	95.5	89.6	-	93.2	71.7
Jaderberg и др.	97.1	92.7	-	95.4	80.7
Baoguang Shi и др. (CRNN) [4]	97.6	94.4	78.2	96.4	80.8
R^2AM	96.8	94.4	78.4	-	80.7
CA-FCN	98.9	99.8	92.0	-	82.1
FACLSTM	99.5	98.6	90.5	-	82.2
Предлагаемый подход	96.7	93.2	74.0	95.3	76.2

ЗАКЛЮЧЕНИЕ

В работе проведен обзор и анализ существующих методов локализации и распознавания текста на изображениях сложных графических сцен. Предложен подход к распознаванию локализованного текста, базирующийся на сочетании рекуррентных, сверточных нейронных сетей и алгоритма CTC-loss. Данная архитектура нейронных сетей реализована при помощи языка программирования Python с применением библиотеки Tensorflow. Проведен эксперимент на двух наборах данных, по результатам которого были построены графики изменения в течение обучения функции потерь, расстояния Левенштейна и точности распознавания на тренировочном и двух тестовых наборах данных. Составлена таблица значений точности распознавания текста. Распознавание проводилось двумя способами: с использованием дополнительного словаря фиксированного размера и без его использования. Для набора данных IIIT5K использовались 2 словаря – размерами 50 и 1000 слов, для набора данных SVT использовался словарь размеров в 50 слов.

По итогам эксперимента на наборе данных SVT без использования словаря была достигнута точность 76.2%, с использованием словаря размером в 50 слов – 95.3%. На наборе данных IIIT5K без словаря была достигнута точность 74.0%, со словарем размером в 50 слов – 96.7%, со словарем размером в 1000 слов – 93.2. Проведено сравнение с результатами других работ, разработанная система показывает результаты, близкие к state-of-the-art решениям с меньшими вычислительными затратами благодаря использованию дополнительных слоев нормализации по мини-батчам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Адрианов А.И. Локализация текста на изображениях сложных графических сцен // Современные проблемы науки и образования. – № 3. – 2013.

2 Y. Kunishige, F. Yaokai, S. Uchida. Scenery Character Detection with Environmental Context // The 11th International Conference on Document Analysis and Recognition (ICDAR), 2011. pp. 1049 – 1053.

3 S. Uchida, Y. Shigeyoshi, Y. Kunishige, F. Yaokai. A Keypoint-Based Approach Toward Scenery Character Detection // The 11th International Conference on Document Analysis and Recognition (ICDAR), 2011. pp. 819 – 823.

4 Y. Du, H. Ai, S. Lao. Dot Text Detection Based on FAST Points // The 11th International Conference on Document Analysis and Recognition (ICDAR), 2011. pp. 435 – 439.

5 A. Coates, B. Carpenter, C. Case, S. Satheesh, B. Suresh, T. Wang, D. Wu, A. Ng. Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning // The 11th International Conference on Document Analysis and Recognition (ICDAR), 2011. pp. 440 – 445.

6 B. Epshtein, E. Ofek, Y. Wexler, Detecting Text in Natural Scenes with Stroke Width Transform // 23rd IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol.V. San Francisco, 2010.

7 Tong He, Weilin Huang, Yu Qiao, Jian Yao. Text-attentional convolutional neural network for scene text detection // IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

8 Pengyuan Lyu, Minghui Liao, Cong Yao, Wenhao Wu, Xiang Bai. Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes. Huazhong University of Science and Technology, 2018.

9 Bissacco, A., Cummins, M., Netzer, Y., Neven, H.: Photoocr: Reading text in uncontrolled conditions // Proc. ICCV, 2013. pp. 785–792.

10 Jaderberg, M., Vedaldi, A., Zisserman, A. Deep features for text spotting // Proc. ECCV, 2014. pp. 512–528.

11 Jaderberg, M., Simonyan, K., Vedaldi, A., Zisserman, A.: Synthetic data and artificial neural networks for natural scene text recognition. CoRR, 2014.

12 Max Jaderberg, Karen Simonyan, Andrea Vedaldi, Andrew Zisserman. Reading Text in the Wild with Convolutional Neural Networks. // International Journal of Computer Vision. – Hingham, MA, USA, 2016. – pp. 1-20.

13 I. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, V.Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks // arXiv:1312.6082, 2014.

14 Shi, B., Bai, X., Yao, C. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition // IEEE Trans. Pattern Anal. Mach. Intell. 39(11), 2017. pp. 2298–2304.

15 Alex Graves, Santiago Fernandez, Faustino Gomez, Jurgen Schmidhuber. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. // ICML '06 Proceedings of the 23rd international conference on Machine learning. – Pittsburgh, Pennsylvania, 2006. – pp. 369-376.

16 S. Ghosh, E. Valveny, A. Bagdanov. Visual attention models for scene text recognition // 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), 2017.

17 F. Yin, Y. Wu, X. Zhang, C. Liu. Scene Text Recognition with Sliding Convolutional Character Models // arXiv:1709.01727, 2017.

18 Y. Wu, F. Yin, X. Zhang, L. Liu, C. Liu. SCAN: Sliding Convolutional Attention Network for Scene Text Recognition // ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 2019.

19 P. Wang, L. Yang, H. Li, Y. Deng, C. Shen, Y. Zhang. A Simple and Robust Convolutional-Attention Network for Irregular Text Recognition // arXiv:1904.01375, 2019.

20 W. Liu, C. Chen, K. Wong. SAFE: Scale Aware Feature Encoder for Scene Text Recognition // Asian Conference on Computer Vision (ACCV), 2018.

21 S. Ioffe, C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning, 2015.

22 Sheng Qian, Hua Liu, Cheng Liu, Si Wu, Hau San Wong. Adaptive activation functions in convolutional neural networks // Neurocomputing, Volume 272, 2018. – pp. 204-212. 16 Kohonen, T. (1988), Learning Vector Quantization, Neural Networks, 1 (suppl 1), 303.

23 S. Hochreiter, Jurgen Schmidhuber. Long short-term memory // Neural Computation, 1997. – pp. 1735–1780.

24 Baoguang Shi, Xiang Bai, Cong Yao. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition // IEEE Transactions on Pattern Analysis and Machine Intelligence. – University of Toronto, Canada, 2015. – pp. 99.

25 Профессиональный информационно-аналитический ресурс, посвя-щенный машинному обучению, распознаванию образов и интеллектуальному анализу данных. [Электронный ресурс]: режим доступа - http://www.machinelearning.ru.

26 Многослойные нейронные сети [Электронный ресурс] : Материал из Википедии — свободной энциклопедии : Версия 89620593, сохранённая в 18:53 UTC 12 декабря 2017 / Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2017. — Режим доступа: http://ru.wikipedia.org/?oldid=89620593 wikipedia.org

27 Bottou, Leon. (2011). From Machine Learning to Machine Reasoning. Computing Research Repository - CORR. 94. . 10.1007/s10994-013-5335-x.

28 Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика = Neural Computing. Theory and Practice. — М.: Мир, 1992. — 240 с. — ISBN 5-03-002115-9.

29 Медведев В.С. Нейронные сети / В.С. Медведев, В.Г. Потемкин – М.: Диалог МИФИ, 2002.

30 Carlos Affonso, Andre Luis Debiasso Deep Learning for biological image classification – Expert Systems with Applications, Volume 85, 2017, pp. 114-122.

31 Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions”. arXiv:1409.4842 2014

32 C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, “Rethinking the Inception Architecture for Computer Vision”. arXiv:1512.00567 2015

33 Статистическая обработка данных [Электронный ресурс] – Режим доступа: http://www.intuit.ru/studies/courses/3632/874/info.

34 Классические методы статистики:метод главных компонент[Электронный ресурс] – Режим доступа: http://r-analytics.blogspot.ru/2012/08/blog-post.html.

35 Профессиональный информационно-аналитический ресурс, посвященный машинному обучению, распознаванию образов и интеллектуальному анализу данных. [Электронный ресурс] – Режим доступа: http://www.machinelearning.ru/

36 Статистический анализ медицинских данных. Применение пакета прикладных программ STATISTICA / Реброва О.Ю. – Москва: Медиа Сфера, 2002. – 312 с.

37 Статистические методы построения эмпирических формул. / Львовский Е.Н. – Москва: Высшая школа, 19 . 239 с.

38 Нейронная сеть Кохонена [Электронный ресурс] : Материал из Википедии — свободной энциклопедии : Версия 89620593, сохранённая в 18:53 UTC 12 декабря 2017 / Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2017. — Режим доступа: http://ru.wikipedia.org/?oldid=89620593 wikipedia.org.

39 Kohonen, T. (1988), Learning Vector Quantization, Neural Networks, 1 (suppl 1), 303.

40 Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1(4):541-551, Winter 1989.

41 Jain, V. and Seung, S. H. (2008). Natural image denoising with convolutional networks. In NIPS’2008.

42 Graham, Benjamin (2014-12-18), "Fractional Max-Pooling", arXiv:1412.6071 [cs.CV].

43 Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009a). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In ICML’2009.

44 Zeiler, M., Krishnan, D., Taylor, G., and Fergus, R. (2010). Deconvolutional networks. In CVPR’2010.

45 Галушкин А. И. Синтез многослойных систем распознавания образов. — М.: Энергия, 1974.

46 Martinetz T. M., Berkovich S. G., Schulten K. J. Neural-gas network for vector quantization and its application to time-series prediction // IEEE Trans. on Neural Networks, 1993, No. 4. — P. 558—569.

47 Linda G. Shapiro and George C. Stockman (2001): «Computer Vision», pp 279—325, New Jersey, Prentice-Hall, ISBN 0-13-030796-3.

48 Ивченко Г. И., Медведев Ю. И. Введение в математическую статистику. — М. : Издательство ЛКИ, 2010. — §2.2. Выборочные моменты: точная и асимптотическая теория. — ISBN 978-5-382-01013-7.

49 Гилл Ф., Мюррей У., Райт М. Практическая оптимизация = Practical Optimization. — М.: Мир, 1985.

50 Горбань А. Н. Обучение нейронных сетей. — М.: СП ПараГраф, 1990.

51 Masakazu Iwamura. Advances of Scene Text Datasets // Department of Computer Science and Intelligent Systems Graduate School of Engineering, Osaka Prefecture University, 2018.

ПРИЛОЖЕНИЕ А

Исходный код разработанной программы на ЭВМ

config.py

```
char_vector = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ-'.!?,\""
num_classes = len(char_vector) + 1
batch_size = 64
max_word_len = 16
img_w = 100
img_h = 31
```

utils.py

```
import numpy as np
import config

def label_to_array(label):
    return [config.char_vector.index(x) for x in label]

def ground_truth_to_word(ground_truth):

    try:

        ground_truth_int = ground_truth.astype(np.int16)
        word = ''.join([config.char_vector[int(i)] for i in ground_truth_int if i in
np.arange(len(config.char_vector))])

    except TypeError:

        print(ground_truth)
        print(ground_truth_int)

    return word

def batch_ground_truth_to_word(batch_ground_truth):

    batch_words = np.empty(config.batch_size, dtype=np.object)

    try:
        for j in np.arange(config.batch_size):
```

```
            batch_words[j] = ''.join([config.char_vector[int(i)] for i in batch_ground_truth[j]
if i in np.arange(len(config.char_vector))])

    except TypeError:
        print(batch_ground_truth)
        print(batch_ground_truth[0])

    return batch_words

def to_max_len(word):
    while len(word) < config.max_word_len:
        word += '-'
    return word

def sparse_tuple_from(sequences, dtype=np.int32):

    indices = []
    values = []

    for n, seq in enumerate(sequences):
        indices.extend(zip([n]*len(seq), [i for i in range(len(seq))]))
        values.extend(seq)

    indices = np.asarray(indices, dtype=np.int64)
    values = np.asarray(values, dtype=dtype)
    shape = np.asarray([len(sequences), np.asarray(indices).max(0)[1]+1],
dtype=np.int64)

    return indices, values, shape
```

data.py

```
import numpy as np
import pandas as pd
from imageio import imread
from scipy.misc import imresize
from utils import label_to_array, to_max_len, sparse_tuple_from
from tensorflow.keras.utils import to_categorical
from mat4py import loadmat
import xmltodict
import random
import config
import tensorflow as tf
```

```
def synth_word_generator(path='90kDICT32px/', mode='train'):

    if mode == 'train':
        file_path = 'annotation_train.txt'
    if mode == 'val':
        file_path = 'annotation_val.txt'
    if mode == 'test':
        file_path = 'annotation_test.txt'

    with open(path + file_path, 'r') as file:
        file_raws = file.readlines()
    dataset_len = len(file_raws)
    j = 0

    while True:

        j += 1

        # Random files for this batch
        batch_raws = [random.choice(file_raws) for _ in np.arange(config.batch_size)]
        x_batch = np.empty([config.batch_size, config.img_h, config.img_w, 1])
        y_batch = np.empty(config.batch_size, dtype=np.object)
        dt_batch = np.empty(config.batch_size, dtype=np.object)

        for i, raw in enumerate(batch_raws):

            try:

                img_path = raw.split(' ')[0]
                label = img_path.split('_')[1].upper()
                target = label_to_array(label)
                img = imread(path + img_path)[:, :, 0]

            except (IndexError, SyntaxError, ValueError, OSError):

                raw = random.choice(file_raws)
                img_path = raw.split(' ')[0]
                label = img_path.split('_')[1].upper()
                target = label_to_array(label)
                img = imread(path + img_path)[:, :, 0]

            img = imresize(img, (config.img_h, config.img_w))[:, :, np.newaxis]

            x_batch[i, :, :, :] = img
```

```
            y_batch[i] = label
            dt_batch[i] = target

        x_batch /= 255
        dt_batch = sparse_tuple_from(np.reshape(np.array(dt_batch), (-1)))

        yield x_batch, y_batch, dt_batch

def svt_word_generator(path='SVT/'):

    xml_path = 'test.xml'

    with open(path + xml_path, 'rb') as file:

        xmlDict = xmltodict.parse(file)
        df = pd.DataFrame.from_dict(xmlDict)

    x_dataset = []
    y_dataset = []
    tg_dataset = []
    lex_dataset = []

    for image in df['tagset']['image']:

        img_path = image['imageName']
        lexicon = image['lex']
        full_img = imread(path + img_path)

        if isinstance(image['taggedRectangles']['taggedRectangle'], list):

            for word in image['taggedRectangles']['taggedRectangle']:

                label = word['tag']
                target = label_to_array(label)
                height = int(word['@height'])
                width = int(word['@width'])

                x = int(word['@x'])
                y = int(word['@y'])
                word = full_img[y:y + height, x:x + width, :]

                if (word.shape[0] == 0) or (word.shape[1] == 0):
                    continue
```

```
                word = imresize(word, (config.img_h, config.img_w))
                x_dataset.append(word)
                y_dataset.append(label)
                tg_dataset.append(target)
                lex_dataset.append(lexicon)

        else:

            word = image['taggedRectangles']['taggedRectangle']
            label = word['tag']
            target = label_to_array(label)
            height = int(word['@height'])
            width = int(word['@width'])
            x = int(word['@x'])
            y = int(word['@y'])
            word = full_img[y:y + height, x:x + width, :]
            word = imresize(word, (config.img_h, config.img_w))

            x_dataset.append(word)
            y_dataset.append(label)
            tg_dataset.append(target)
            lex_dataset.append(lexicon)

    x_dataset = np.mean(np.array(x_dataset, dtype=np.float32), axis=-1)[:, :, :,
np.newaxis]
    x_dataset = np.reshape(x_dataset[:-6], [10, 64, 31, 100, 1])
    x_dataset /= 255
    y_dataset = np.array(y_dataset, dtype=np.object)
    y_dataset = np.reshape(y_dataset[:-6], [10, 64])
    tg_dataset = np.array(tg_dataset, dtype=np.object)
    tg_dataset = np.reshape(tg_dataset[:-6], [10, 64])
    lex_dataset = np.array(lex_dataset, dtype=np.object)
    lex_dataset = np.reshape(lex_dataset[:-6], [10, 64])

    for i in np.arange(10):

        x_batch = x_dataset[i]
        y_batch = y_dataset[i]
        tg_batch = tg_dataset[i]
        lex_batch = lex_dataset[i]
        tg_batch = sparse_tuple_from(np.reshape(np.array(tg_batch), (-1)))

        yield x_batch, y_batch, tg_batch, lex_batch
```

```
def IIIT5K_word_generator(path='IIIT5K/'):

    file_path = 'testdata.mat'
    key = 'testdata'

    x_dataset = []
    y_dataset = []
    tg_dataset = []
    lex50_dataset = []
    lex1000_dataset = []

    file = loadmat(path + file_path)
    dataset_len = len(file[key]['ImgName'])

    for i in np.arange(dataset_len):

        img_path = file[key]['ImgName'][i]
        label = file[key]['GroundTruth'][i]
        lexicon_50 = file[key]['smallLexi'][i]
        lexicon_1000 = file[key]['mediumLexi'][i]
        target = label_to_array(label)

        img = imread(path + img_path)
        if img.ndim == 2:
            img = np.stack([img, img, img], axis=2)
        img = imresize(img, (config.img_h, config.img_w))

        x_dataset.append(img)
        y_dataset.append(label)
        tg_dataset.append(target)
        lex50_dataset.append(lexicon_50)
        lex1000_dataset.append(lexicon_1000)

    x_dataset = np.mean(np.array(x_dataset, dtype=np.float32), axis=-1)[:, :, :,
np.newaxis]
    x_dataset = np.reshape(x_dataset[:-56], [46, 64, 31, 100, 1])
    x_dataset /= 255
    y_dataset = np.array(y_dataset, dtype=np.object)
    y_dataset = np.reshape(y_dataset[:-56], [46, 64])
    tg_dataset = np.array(tg_dataset, dtype=np.object)
    tg_dataset = np.reshape(tg_dataset[:-56], [46, 64])
    lex50_dataset = np.array(lex50_dataset, dtype=np.object)
    lex50_dataset = np.reshape(lex50_dataset[:-56], [46, 64])
    lex1000_dataset = np.array(lex1000_dataset, dtype=np.object)
```

```
    lex1000_dataset = np.reshape(lex1000_dataset[:-56], [46, 64, 1000])

    for i in np.arange(46):

        x_batch = x_dataset[i]
        y_batch = y_dataset[i]
        tg_batch = tg_dataset[i]
        tg_batch = sparse_tuple_from(np.reshape(np.array(tg_batch), -1))
        lex50_batch = lex50_dataset[i]
        lex1000_batch = lex1000_dataset[i]

        yield x_batch, y_batch, tg_batch, lex50_batch, lex1000_batch
```

train_model.py

```
import numpy as np
import tensorflow as tf
from tensorflow.contrib import rnn
from utils import ground_truth_to_word, batch_ground_truth_to_word
from data import synth_word_generator, svt_word_generator, IIIT5K_word_generator
import config

save_path = '.'
train_len = 7224612
train_epoch_len = train_len // config.batch_size
iteration_count = train_epoch_len * 5

inputs = tf.placeholder(tf.float32, [config.batch_size, config.img_h, config.img_w, 1],
name='inputs')
targets = tf.sparse_placeholder(tf.int32, name='targets')
seq_len = tf.placeholder(tf.int32, [None], name='seq_len')

# Feature extraction
conv1 = tf.layers.conv2d(inputs=inputs, filters = 64, kernel_size = (3, 3), padding =
"same", activation=tf.nn.relu, name='conv_1')
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2,
name='pool_1')
conv2 = tf.layers.conv2d(inputs=pool1, filters = 128, kernel_size = (3, 3), padding =
"same", activation=tf.nn.relu, name='conv_2')
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2,
name='pool_2')
conv3 = tf.layers.conv2d(inputs=pool2, filters = 256, kernel_size = (3, 3), padding =
"same", activation=tf.nn.relu, name='conv_3')
```

```
bnorm1 = tf.layers.batch_normalization(conv3, name='bn_1')
conv4 = tf.layers.conv2d(inputs=bnorm1, filters = 256, kernel_size = (3, 3), padding =
"same", activation=tf.nn.relu, name='conv_4')
pool3 = tf.layers.max_pooling2d(inputs=conv4, pool_size=[2, 2], strides=[1, 2],
padding="same", name='pool_3')
conv5 = tf.layers.conv2d(inputs=pool3, filters = 512, kernel_size = (3, 3), padding =
"same", activation=tf.nn.relu, name='conv_5')

bnorm2 = tf.layers.batch_normalization(conv5, name='bn_2')
conv6 = tf.layers.conv2d(inputs=bnorm2, filters = 512, kernel_size = (3, 3), padding =
"same", activation=tf.nn.relu, name='conv_6')
pool4 = tf.layers.max_pooling2d(inputs=conv6, pool_size=[2, 2], strides=[1, 2],
padding="same", name='pool_4')
conv7 = tf.layers.conv2d(inputs=pool4, filters = 512, kernel_size = (2, 2), padding =
"valid", activation=tf.nn.relu, name='conv_7')

# Reshape for LSTM and max char count
reshaped_cnn_output = tf.reshape(conv7, [config.batch_size, -1, 512],
name='reshaped_cnn')
max_char_count = reshaped_cnn_output.get_shape().as_list()[1]
max_char_count_name = tf.constant(max_char_count, name='max_char_count')
print('='*20, max_char_count)

with tf.variable_scope(None, default_name="bidirectional-rnn-1"):

    # Forward
    lstm_fw_cell_1 = rnn.BasicLSTMCell(256)
    # Backward
    lstm_bw_cell_1 = rnn.BasicLSTMCell(256)
    # First bidirectional LSTM
    inter_output, _ = tf.nn.bidirectional_dynamic_rnn(lstm_fw_cell_1, lstm_bw_cell_1,
reshaped_cnn_output, seq_len, dtype=tf.float32)
    inter_output = tf.concat(inter_output, 2)

with tf.variable_scope(None, default_name="bidirectional-rnn-2"):

    # Forward
    lstm_fw_cell_2 = rnn.BasicLSTMCell(256)
    # Backward
    lstm_bw_cell_2 = rnn.BasicLSTMCell(256)
    # Second bidirectional LSTM
    lstm_outputs, _ = tf.nn.bidirectional_dynamic_rnn(lstm_fw_cell_2, lstm_bw_cell_2,
inter_output, seq_len, dtype=tf.float32)
    lstm_outputs = tf.concat(lstm_outputs, 2)
```

```
logits = tf.reshape(lstm_outputs, [-1, 512], name='reshaped_lstm')

W = tf.Variable(tf.truncated_normal([512, config.num_classes], stddev=0.1),
name="W")
b = tf.Variable(tf.constant(0., shape=[config.num_classes]), name="b")

logits = tf.matmul(logits, W) + b
logits = tf.reshape(logits, [config.batch_size, -1, config.num_classes], name='logits')

# Final layer, the output of the BLSTM
logits = tf.transpose(logits, (1, 0, 2), name='transpose_logits')

# Loss and cost calculation
loss = tf.nn.ctc_loss(targets, logits, seq_len)
cost = tf.reduce_mean(loss, name='cost')

optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)

decoded, log_prob = tf.nn.ctc_beam_search_decoder(logits, seq_len,
merge_repeated=False)
dense_decoded = tf.sparse_tensor_to_dense(decoded[0], default_value=-1,
name='dense_decoded')

acc = tf.reduce_mean(tf.edit_distance(tf.cast(decoded[0], tf.int32), targets),
name='accuracy')

with tf.Session() as sess:

    init = tf.global_variables_initializer()
    sess.run(init)

    synth_train_gen = synth_word_generator('90kDICT32px/', mode='train')
    svt_gen = svt_word_generator(path='SVT/')
    IIIT5K_gen = IIIT5K_word_generator(path='IIIT5K/')

    train_loss_hist = []
    val_loss_hist = []
    train_acc_hist = []
    val_acc_hist = []
    svt_loss_hist = []
    svt_acc_hist = []
    IIIT5K_loss_hist = []
    IIIT5K_acc_hist = []
```

```
    train_wer_hist = []
    val_wer_hist = []
    svt_wer_hist = []
    IIIT5K_wer_hist = []

    # Train
    for i in np.arange(iteration_count):

        x_batch, y_batch, dt_batch = synth_train_gen.__next__()
        op, decoded_value, loss_value, acc_value = sess.run([optimizer, dense_decoded,
cost, acc], feed_dict={inputs: x_batch, targets: dt_batch, seq_len: [max_char_count] *
config.batch_size})

        if i % 50 == 0:

            print('True synth train:\t{}'.format(y_batch[0]))
            print('Pred synth train:\t{}'.format(ground_truth_to_word(decoded_value[0])))
            print('[{}] Synth 90k iteration. Train loss: {}\tTrain accuracy: {}'.format(i,
loss_value, acc_value))
            train_wer = np.sum(np.equal(y_batch,
batch_ground_truth_to_word(decoded_value)))/config.batch_size
            print('Train WER:\t{}'.format(train_wer))

            train_wer_hist.append(train_wer)
            train_loss_hist.append(loss_value)
            train_acc_hist.append(acc_value)

            x_batch_val, y_batch_val, dt_batch_val = synth_val_gen.__next__()
            val_decoded_value, val_loss_value, val_acc_value = sess.run([dense_decoded,
cost, acc], feed_dict={inputs: x_batch_val, targets: dt_batch_val, seq_len:
[max_char_count] * config.batch_size})

            print('True synth val:\t{}'.format(y_batch_val[0]))
            print('Pred synth
val:\t{}'.format(ground_truth_to_word(val_decoded_value[0])))
            print('[{}] Synth 90k iteration. Val loss: {}\tVal accuracy: {}'.format(i,
val_loss_value, val_acc_value))
            val_wer = np.sum(np.equal(y_batch_val,
batch_ground_truth_to_word(val_decoded_value)))/config.batch_size
            print('Val WER:\t{}'.format(val_wer))

            val_wer_hist.append(val_wer)
            val_loss_hist.append(val_loss_value)
            val_acc_hist.append(val_acc_value)
```

```
            x_batch_svt, y_batch_svt, dt_batch_svt = svt_gen.__next__()
            svt_decoded_value, svt_loss_value, svt_acc_value, _, _ =
sess.run([dense_decoded, cost, acc], feed_dict={inputs: x_batch_svt, targets:
dt_batch_svt, seq_len: [max_char_count] * config.batch_size})

            print('True svt:\t{}'.format(y_batch_svt[0]))
            print('Pred svt:\t{}'.format(ground_truth_to_word(svt_decoded_value[0])))
            print('[{}] SVT iteration. Test loss: {}\tTest accuracy: {}'.format(i,
svt_loss_value, svt_acc_value))
            svt_wer = np.sum(np.equal(y_batch_svt,
batch_ground_truth_to_word(svt_decoded_value)))/config.batch_size
            print('svt WER:\t{}'.format(svt_wer))

            svt_wer_hist.append(svt_wer)
            svt_loss_hist.append(svt_loss_value)
            svt_acc_hist.append(svt_acc_value)

            x_batch_IIIT5K, y_batch_IIIT5K, dt_batch_IIIT5K = IIIT5K_gen.__next__()
            IIIT5K_decoded_value, IIIT5K_loss_value, IIIT5K_acc_value, _, _ =
sess.run([dense_decoded, cost, acc], feed_dict={inputs: x_batch_IIIT5K, targets:
dt_batch_IIIT5K, seq_len: [max_char_count] * config.batch_size})

            print('True IIIT5K:\t{}'.format(y_batch_IIIT5K[0]))
            print('Pred
IIIT5K:\t{}'.format(ground_truth_to_word(IIIT5K_decoded_value[0])))
            print('[{}] IIIT5K iteration. Test loss: {}\tTest accuracy: {}'.format(i,
IIIT5K_loss_value, IIIT5K_acc_value))
            IIIT5K_wer = np.sum(np.equal(y_batch_IIIT5K,
batch_ground_truth_to_word(IIIT5K_decoded_value)))/config.batch_size
            print('IIIT5K WER:\t{}'.format(IIIT5K_wer))

            IIIT5K_wer_hist.append(IIIT5K_wer)
            IIIT5K_loss_hist.append(IIIT5K_loss_value)
            IIIT5K_acc_hist.append(IIIT5K_acc_value)

            if i % 5000 == 0:

                saver = tf.train.Saver(tf.global_variables(), max_to_keep=10)
                saver.save(sess, save_path)
```

evaluate.py

```
import numpy as np
```

```
import tensorflow as tf
from utils import ground_truth_to_word, batch_ground_truth_to_word
from test_data import svt_word_generator, IIIT5K_word_generator
from editdistance import distance
import config

def get_predict_gen(dataset='SVT'):

    with tf.Session() as sess:

        # Load model
        saver = tf.train.import_meta_graph('..meta')
        saver.restore(sess, tf.train.latest_checkpoint('./'))
        graph = tf.get_default_graph()

        # Get tensors
        inputs = graph.get_tensor_by_name("inputs:0")
        targets_shape = graph.get_tensor_by_name("targets/shape:0")
        targets_values = graph.get_tensor_by_name("targets/values:0")
        targets_indices = graph.get_tensor_by_name("targets/indices:0")
        targets = tf.SparseTensor(targets_indices, targets_values, targets_shape)
        seq_len = graph.get_tensor_by_name("seq_len:0")
        max_char_count = graph.get_tensor_by_name("reshaped_cnn:0")
.get_shape().as_list()[1]
        acc = graph.get_tensor_by_name("accuracy:0")
        dense_decoded = graph.get_tensor_by_name("dense_decoded:0")

        if dataset == 'SVT':

            dataset_gen = svt_word_generator(path='SVT/')

            for i in np.arange(10):

                x_batch_svt, y_batch_svt, dt_batch_svt, lex_batch_svt =
dataset_gen.__next__()
                acc_value, svt_decoded_value = sess.run([acc, dense_decoded],
feed_dict={inputs: x_batch_svt, targets: dt_batch_svt, seq_len: [max_char_count] *
config.batch_size})

                yield svt_acc_value, batch_ground_truth_to_word(svt_decoded_value),
y_batch_svt, lex_batch_svt

        if dataset == 'IIIT5K':
```

```
        dataset_gen = IIIT5K_word_generator(path='IIIT5K/')

        for i in np.arange(46):

            x_batch_IIIT5K, y_batch_IIIT5K, dt_batch_IIIT5K, lex50_batch_IIIT5K,
lex1000_batch_IIIT5K = dataset_gen.__next__()
            IIIT5K_acc_value, IIIT5K_decoded_value = sess.run([acc, dense_decoded],
feed_dict={inputs: x_batch_IIIT5K, targets: dt_batch_IIIT5K, seq_len:
[max_char_count] * config.batch_size})

            yield IIIT5K_acc_value,
batch_ground_truth_to_word(IIIT5K_decoded_value), y_batch_IIIT5K,
lex50_batch_IIIT5K, lex1000_batch_IIIT5K

if __name__ == '__main__':

    svt_gen = get_predict_gen(dataset='SVT')
    IIIT5K_gen = get_predict_gen(dataset='IIIT5K')
    total_svt_acc = []
    total_svt_acc_d50 = []
    total_IIIT5K_acc_d50 = []
    total_IIIT5K_acc_d1000 = []
    total_IIIT5K_acc = []

    for i in np.arange(10):

        svt_acc, svt_pred, svt_true, lexicon = next(svt_gen)
        svt_found_words = []

        for j in np.arange(config.batch_size):

            current_lex = lexicon[j].split(',')
            pred_word = str(svt_pred[j])
            distances = [distance(pred_word, lex_word) for lex_word in current_lex]
            found_word = current_lex[np.argmin(distances)]
            svt_found_words.append(found_word)
            total_svt_acc.append(np.sum(np.equal(svt_true, svt_pred))/config.batch_size)
            total_svt_acc_d50.append(np.sum(np.equal(svt_true,
svt_found_words))/config.batch_size)

        print('Total SVT accuracy\n', np.mean(total_svt_acc))
        print('Total SVT accuracy with 50 words dict', np.mean(total_svt_acc_d50))

        for i in np.arange(46):
```

```
        IIIT5K_acc, IIIT5K_pred, IIIT5K_true, IIIT5K_lex50, IIIT5K_lex1000 =
next(IIIT5K_gen)
        IIIT5K_found_words50 = []
        IIIT5K_found_words1000 = []

        for j in np.arange(config.batch_size):

            current_lex50 = IIIT5K_lex50[j]
            current_lex1000 = IIIT5K_lex1000[j]
            pred_word = str(IIIT5K_pred[j])
            distances50 = [distance(pred_word, lex_word) for lex_word in current_lex50]
            distances1000 = [distance(pred_word, lex_word) for lex_word in
current_lex1000]
            found_word50 = current_lex50[np.argmin(distances50)]
            found_word1000 = current_lex1000[np.argmin(distances1000)]
            IIIT5K_found_words50.append(found_word50)
            IIIT5K_found_words1000.append(found_word1000)

            total_IIIT5K_acc.append(np.sum(np.equal(IIIT5K_true,
IIIT5K_pred))/config.batch_size)
            total_IIIT5K_acc_d50.append(np.sum(np.equal(IIIT5K_true,
IIIT5K_found_words50))/config.batch_size)
            total_IIIT5K_acc_d1000.append(np.sum(np.equal(IIIT5K_true,
IIIT5K_found_words1000))/config.batch_size)

        print('Total IIIT5K accuracy\n', np.mean(total_IIIT5K_acc))
        print('Total IIIT5K accuracy with 50 words dict\n',
np.mean(total_IIIT5K_acc_d50))
        print('Total IIIT5K accuracy with 1000 words dict\n',
np.mean(total_IIIT5K_acc_d1000))
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Системы искусственного интеллекта
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Г.М. Цибульский
подпись инициалы, фамилия
«___» _____ 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

«Применение искусственных нейронных сетей к задаче распознавания текста на изображениях сложных графических сцен»
тема

09.04.01 «Информатика и вычислительная техника»
код и наименование направления

09.04.01.10 «Интеллектуальные информационные системы»
код и наименование магистерской программы

Научный руководитель _____ 29.06.19 канд.техн.наук, доцент А.В. Пятаева
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник _____ 29.06.19 С.А. Генза
подпись, дата инициалы, фамилия

Рецензент _____ 29.06.19 канд.техн.наук, доцент В.В. Вдовенко
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2019